USB TRANSCEIVER

Oleh :

NOR WANY AFFINDA BT MHD POSTOTDIN AFANDI

WEK 010 365

Penyelia : En. Mohd Yamani Idna Bin Idris

Moderator : En. NoorZaily Bin Mohamed Noor

Isi Kandungan

Abstrak..... ii

Penghargaan..... iii-iv

Bab 1 : Pengenalan..... 1

1.1 Pengenalan pada tajuk..... 1

1.2 Definisi Masalah..... 1

1.3 Skop..... 2

1.4 Objektif / Tujuan..... 3

1.5 Kekangan..... 4

1.6 Penjadualan..... 5

1.6.1 Jadual Pembangunan Sistem..... 6

Bab 2 : Kajian Literasi..... 7

2.1 Pengenalan..... 7

2.2 Sejarah USB..... 7-8

2.3 Senibina.....9

2.3 Rajah Senibina.....10

2.3.1 Rajah.....11-12

2.4 Mengapa Menggunakan USB..... 13-14

Bab 3: Metodologi

3.1 Pengenalan Metodologi..... 15

3.2 Rekabentuk Asas Metodologi..... 16

Bab 4: Analisis Sistem..... 17

4.1 Sejarah VHDL..... 17

4.2 Latar belakang..... 17

4.3 Apa itu VHDL..... 18-19

4.4 Kelebihan VHDL..... 19-21

Bab 5 : Rekabentuk Sistem..... 22

5.1 Pengenalan..... 22

Rajah 5.1..... 23

Jadual 5.1..... 24-26

5.2 Cylic Redundancy Check (CRC)..... 27

5.2.1 Contoh Pengiraan CRC..... 27

5.2.2 Dasar Teori CRC..... 28

Rajah 5.2..... 29

5.3 Packet Dissambler (Pd)..... 30

5.4 Digital Phase Locked Loop (dpll)..... 30

Rajah 5.3..... 31

5.5 Format Paket..... 31

5.6 Paket Pengenal (PID)..... 32

5.6.1 Jenis Paket Dalam USB..... 32

5.5 Rajah..... 32

5.7 Format Paket Data..... 33

Bab 6 : Pengenalan..... 1

6.2 Pembangunan modul.....34

6.3 Pembangunan Testbench.....35

6.4 Pengujian Dengan Perisian Peak FPGA..... 35

6.5 Pengkompil..... 36

6.6 Pautan Dan Pemetaan Port.....36

6.7 Simulasi 36

6.1 Rajah Simulasi Bagi Modul Utama CRC..... 37

6.1 Jadual Pernyataan Input Dan Output CRC..... 38

Bab 7 : Perbincangan Dan Kesimpulan.....39

7.1 Pengenalan.....39

7.2 Perubahan Terhadap Rekabentuk.....39

7.2.1 Rekabentuk terbaru Menambah Satu Modul.....40

7.3 Masalah Pembangunan yang Dihadapi.....41

7.3.1 Pengetahuan Tentang VHDL.....41

7.3.2 Had Masa.....42

7.3.3 Sumber Rujukan..... 43

7.3.4 Perkakasan Dan Perisian.....43

7.4 Cadangan Masa Hadapan.....44

7.4.1 Penambahan Modul untuk multiplexer.....44

7.4.2 Menyiapkan testbench untuk semua modul.....45

7.5 Kesimpulan.....45-46

Appendiks A

Appendiks B

Rujukan

ABSTRAK

Pembangunan dalam teknologi komunikasi memerlukan suatu teknologi yang lebih maju dan canggih. Ia mempunyai kaitan dengan projek ini di mana ia melibatkan penghantaran dan penerimaan data melalui USB dengan menggunakan VHDL.

Pada hari ini, kelancaran komunikasi amat penting dan ianya juga amat disokong oleh peranti-peranti yang moden pada masa kini. Contohnya, USB 2.0 merupakan salah satu peranti yang digunakan untuk berkomunikasi di mana USB berperanan di dalam penghantaran dan penerimaan data.

Kewujudan USB telah dapat menyelesaikan beberapa masalah yang timbul dari dulu lagi seperti penyambungan kabel-kabel peranti komputer yang berselirat, perkongsian port sesiri antara pencetak dan modem yang menyebabkan prosesnya perlahan dan bilangan slot kad yang terhad bagi peranti yang memerlukan penyambungan pantas.

Selain itu, USB telah memberikan satu cara tunggal, berpiawai dan mudah digunakan bagi penyambungan kabel dari satu hinggalah 127 peranti ke satu komputer. Ia juga menyediakan *bandwith* tambahan untuk multimedia dan aplikasi storan serta kelajuan transmisi data yang lebih pantas.

PENGHARGAAN

Alhamdulillah, syukur ke hadrat Allah SWT kerana dengan limpah kurnia dan izinNya, dapat juga saya siapkan projek Latihan Ilmiah 1.

Pertama sekali saya ingin mengucapkan jutaan terima kasih yang tak terhingga kepada En. Mohd Yamani Idna Idris selaku penyelia yang banyak membantu saya. Kesungguhan beliau untuk membantu saya sebanyak mungkin serta motivasi yang diberikan amat saya hargai. Selain itu, ucapan terima kasih juga diucapkan buat En. Noor Zaily Mohd Noor di atas segala komen dan teguran yang diberikan.

Sekalung penghargaan juga saya tujukan kepada seluruh ahli keluarga saya terutamanya kedua ibu bapa saya, En. Mohd Postotdin Afandi Bin Mohd Najemuddin dan Puan Jamnah Bt Husin serta adik beradik saya di atas segala kasih sayang, kesabaran serta sentiasa memahami saya sepanjang masa. Saya tak akan dapat lakukan semua ini tanpa mereka.

Ucapan terima kasih ini juga saya ucapkan kepada rakan-rakan seperjuangan yang telah banyak membantu saya terutamanya Nurul, Leen, Zaki dan Wan.

Akhir sekali, buat semua yang terlibat secara langsung atau tidak langsung dalam membantu saya menyiapkan projek ini saya ucapkan terima kasih. Hanya Allah yang dapat membalas jasa baik anda semua.

Sekian Terima kasih.

Nor Wany Affinda Bt Hj. Mhd Postotdin Afandi

(WEK 010 365)

Jabatan Sistem dan Teknologi Komputer

Fakulti Sains Komputer dan Teknlogi Maklumat

Universiti Malaya, Kuala Lumpur.

BAB SATU

PENGENALAN

1. PENGENALAN

1.1 Pengenalan pada tajuk

Pada masa sekarang, kelancaran komunikasi amat bergantung pada pembangunan teknologi seperti yang akan dibangunkan pada projek ini iaitu USB Transceiver. Secara keseluruhan, projek ini adalah mengenai penghantaran dan penerimaan data dalam USB. Walaubagaimana pun, dalam menuju ke era teknologi, kepantasan adalah amat dititikberatkan.

Untuk meningkatkan prestasi, kadar penghantaran data dalam USB, VHDL akan digunakan.

1.2 Definisi Masalah

Adalah amat penting untuk mengenalpasti masalah sama ada yang sudah berlaku atau yang dijangka akan berlaku. Ini adalah bertujuan untuk mencapai objektif pembangunan projek iaitu untuk memperbaiki sistem yang ada sekaligus dapat meningkatkan prestasi sistem. Projek haruslah siap pada tempoh atau jadual kerja yang telah ditetapkan.

1.3 Skop

Skop permasalahan yang perlu diselesaikan di sini terbahagi kepada dua iaitu : -

i- Dari segi perlaksanaan. Sukar untuk mengimplementasikan sistem yang ada kepada perkakasan kerana ia melibatkan pengkodan perkakasan yang mungkin merujuk kepada protokol-protokol yang lain. Sistem yang sedia ada masih belum dapat mengatasi masalah kesesakan data, kawalan terhadap kehilangan data atau data rosak.

ii- Kesesuaian pembangunan rekabentuk terhadap keadaan rangkaian. Kita perlu pastikan ia dapat berkomunikasi dengan perkakasan yang akan dibangunkan dengan baik kerana masalah mungkin berlaku dengan perubahan perlaksanaan daripada perisian ke perkakasan sepenuhnya.

1.3.1 Pembahagian Skop Projek

Memandangkan projek pembangunan USB Transceiver ini ditugaskan kepada 2 orang, maka pembahagian tugas untuk setiap orang adalah mengikut modul-modul bagi memudahkan lagi aktiviti-aktiviti pembangunan. Berikut adalah pembahagian tugasan mengikut modul yag telah dipersetujui oleh kedua-dua belah pihak :

1. **Nor Wany Affinda Bt. Mhd Postotdin Afandi – WEK 010365**

   Modul-modul yang dibangunkan ialah modul **CRC, PD** serta **DPLL**

2. **Nurul Akmal Bt.Othman – WEK 010411**

   Modul-modul yang dibangunkan ialah modul **NRZI** dan **Bit Stuffing.**

1.4 Objektif / Tujuan

i- Mengurangkan kos pembangunan rekabentuk kerana ini perlaksanaan yang hanya tertumpu di dalam perkakasan sahaja yang mana ia boleh meningkatkan kelajuan iaitu tidak ada kos untuk perisisan, pengoperasian dan alatan pembangunan.

ii- Untuk menyelesaikan masalah sistem yang sedia ada iaitu perlaksanaan USB daripada perisian ke dalam bentuk perkakasan dengan meningkatkan prestasi sistem yang boleh menjamin keutuhan data dan kelancaran pemprosesan data ke dalam perkakasan.

iii- Melaksanakan USB Tranceiver dengan mengunakan pendekatan metodologi VHDL. Ini adalah kerana bahasa HDL ini sangat sesuai untuk melaksanakan FPGA kepada teknik ASIC.

iv- Untuk mengenali USB, bagaimana ia berfungsi dan cara ia menghantar dan menerima data.

v- Memahami dengan jelas gambarajah blok dan kod aturcara serta membina sistem berdasarkan gambarajah blok.

1.5 Kekangan

Kegagalan pembangunan projek dalam mencapai objektif boleh dikaitkan dengan masalah kekangan yang berlaku yang disebabkan oleh masa, kos dan kebolehan perisian yang digunakan. Kekangan yang mungkin dihadapi ialah :-

i- Masa untuk membangunkan projek ini adalah terhad. Perlaksanaan rekabentuk ini dijangka sehingga proses simulasi. Ini adalah kerana pembangunan USB Transceiver melibatkan banyak proses dan ini akan memakan masa untuk menyiapkannya.

ii- Kegagalan VHDL bagi memenuhi keperluan fungsian USB Transceiver dari segi pengkodan dan simulasi.

iii- Ralat pada kod sumber VHDL mungkin berlaku ketika kompilasi. Ini mungkin disebabkan oleh keupayaan perpustakaan VHDL yang terhad dan tidak memenuhi sintaks yang digunakan untuk pengkodan USB Transceiver.

1.6 Penjadualan

Bab ini menerangkan secara ringkas penjadualan proses kerja yang dijangka akan siap.

1.6.1 Jadual Pembangunan Sistem

Penjadualan pembangunan projek ini amat penting bagi memastikan semua fasa pembangunan dilaksanakan dalam jangkamasa yang telah ditetapkan dan projek dapat disiapkan mengikut jadual pembangunan projek.

Fasa – fasa yang terlibat ialah :-

- Analisis Keperluan
- Analisis Sistem
- Rekabentuk Sistem
- Pembangunan Modul
- Pengujian Integrasi

USB Transceiver									
		Bulan							
Bil	Fasa	Jun 2004	Julai 2004	Ogos 2004	Sep 2004	Okt 2004	Nov 2004	Dis 2004	Jan 2005
1	Analisis Keperluan								
2	Analisis Sistem								
3	Rekabentuk Sistem								
4	Pembangunan Modul								
5	Pengujian Integrasi								
6	Pengujian Projek								
7	Dokumentasi								

Jadual 1.1 Jadual Pembangunan Sistem

BAB DUA

KAJIAN LITERASI

Bab 2: KAJIAN LITERASI

2.1 Pengenalan

USB merupakan singkatan daripada *Universal Serial Bus*. Ianya merupakan protokol untuk menghantar dan menerima data daripada peranti digital. Ia juga adalah antaramuka piawai untuk menyambungkan peranti yang bersambung dengan komputer (peripheral devices) seperti tetikus, pengimbas, papan kekunci, dan pencetak. Tidak seperti peranti yang lain yang disambungkan ke port, peranti USB boleh digunakan pada komputer tanpa perlu diaktifkan (restart) semula. Terdapat 2 jenis USB iaitu USB 1.0 dan USB 2.0. Kedua-dua USB ini berbeza dari segi kelajuannya di mana USB 2.0 mempunyai kadar pemindahan data sebanyak 480 Mbps iaitu 40 kali lebih laju daripada USB 1.0 yang hanya mempunyai 12 Mbps kelajuan maksimumnya.

2.2 Sejarah USB

Dengan peningkatan penggunaan Komunikasi, Internet dan telefon mudah alih secara amnya, penyambungan komunikasi telah berkembang pada kadar yang tidak dapat dibayangkan pada lima tahun yang lalu. Hasilnya masalah ini wujud akibat industri tanpa wayar 'wireless' yang berhubungkait dengan jalur lebar yang terhad. Secara amnya, piawaian yang terhad kepada kadar kelajuan komunikasi iaitu sebanyak 115000bps di mana ianya tidak memadai dengan teknologi masa kini.

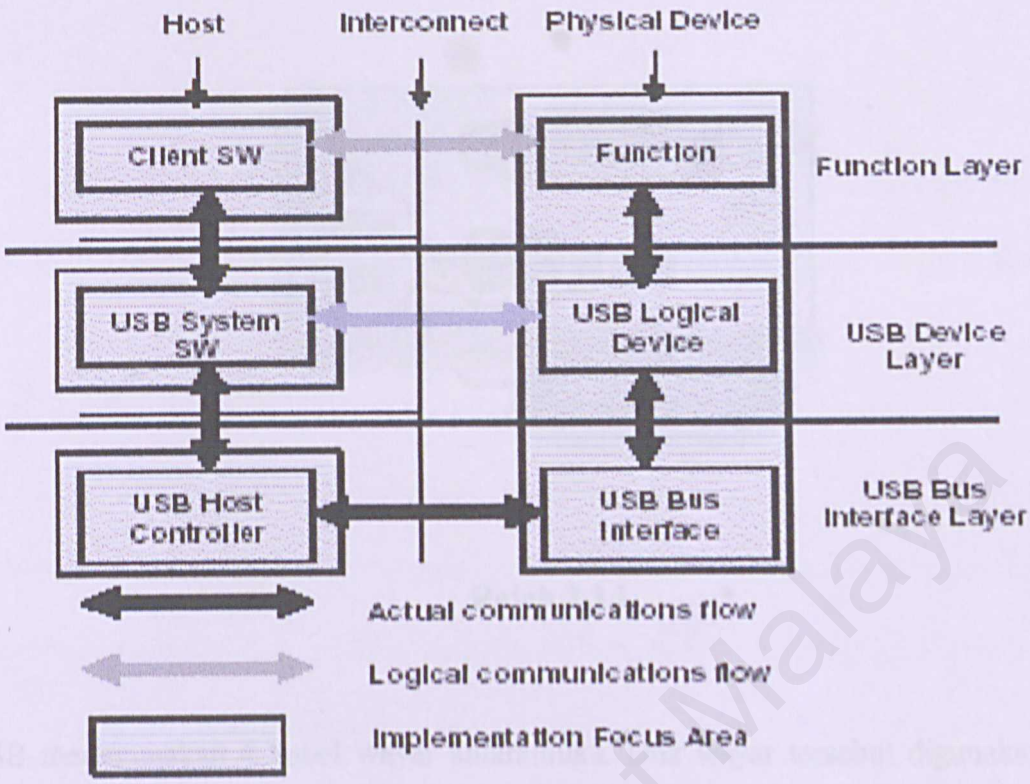
Pada mulanya, USB 1.0 dikeluarkan pada Januari 1996. Ia menyokong 1.5 Megabits per saat (Berkelajuan rendah) dan 12 Megabits per saat bagi kadar pemindahan (berkelajuan tinggi). Peratus kadar data berkaitan dengan limpahan protokol USB. Jadi pemindahan data sebenar kurang dari kelajuan tertentu. Berapa kekurangannya adalah bergantung pada jenis pemindahan dan saiz paket.

USB 1.1 pula dikeluarkan pada bulan September 1998 dan edisi ini banyak mengatasi masalah – masalah yang ada pada USB 1.0.

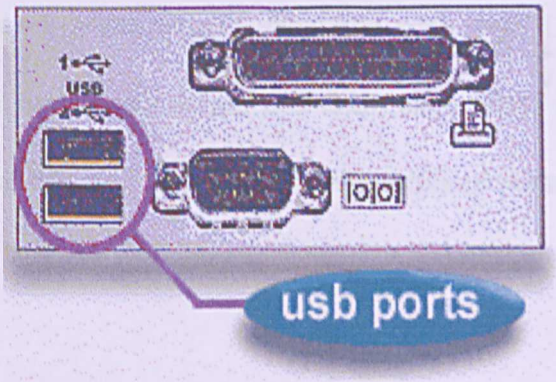
Yang terbaru buat masa ini ialah USB 2.0 di mana ia telah dikeluarkan pada tahun 2000 dengan kelajuannya bertambah hingga 480 megabits per saat. USB 2.0 ini bersesuaian dengan mana-mana USB 1.X. Akan tetapi, terdapat sistem pengoperasian bagi komputer peribadi tidak mempunyai sokongan USB 2.0 sehinggalah pada tahun 2001. Kebanyakkan komputer peribadi telah ada peranti penyambungan USB 2.0 pada akhir 2000.

2.3 Senibina

USB telah dibina daripada pelbagai struktur lapisan. Terdapat beberapa sambungan mudah dari hos kepada peranti yang memerlukan interaksi antara bilangan lapisan dan entiti. Sambungan mudah hos kepada peranti memerlukan interaksi antara bilangan lapisan dan entiti seperti Rajah 2.3. Lapisan antaramuka USB (USB *Interface layer)* menyediakan sambungan fizikal antara hos dan peranti. Lapisan peranti USB (USB *Device Layer)* merupakan pandangan Perisian Sistem USB untuk menunjukkan operasi bersama peranti. *Lapisan Fungsi (Function Layer)* menyediakan kebolehan tambahan kepada hos melalui lapisan perisian klien padanan yang tepat. (appropriate matched client software layer) *Setiap lapisan fungsi dan peranti USB (USB Device and Function Layers)* mempunyai pandangan secara komunikasi logikal melalui lapisan yang sebenarnya dengan menggunakan USB Bus Interface Layer untuk menyelesaikan penghantaran data.



Rajah 2.3



Rajah 2.3.1

USB menggunakan 4 kabel wayar antaramuka. Dua wayar tersebut digunakan dalam mod yang berbeza bagi kedua-dua penghantaran dan penerimaan data, manakala yang dua wayar lagi adalah untuk kuasa dan bumi. Sumber kuasa ke peranti USB boleh dating dari hos, hub atau peranti *"self powered"*.

Terdapat dua jenis penyambung yang berlainan pada setiap hujung kabel USB. Salah satunya adalah untuk komunikasi *upstream* dan yang satu lagi adalah untuk *downstream.* Setiap panjang kabel adalah terhad kapeda 5 meter.

USB mempunyai empat jenis mod pemindahan komunikasi :

- Kawalan / control,
- Sampukan / interrupt,
- bulk, dan
- isochronous.

Mod Kawalan / Control mode adalah disahkan oleh hos. Dalam mod ini, setiap pemindahan data mesti hantar data dalam dua arah, akan tetapi hanya satu arah dalam satu masa . Mod kawalan ini digunakan untuk mengesahkan peranti, tetapi ia boleh juga digunakan juga untuk memindahkan sebilangan kecil data.

Dalam **Mod Sampukan / Interrupt mode,** sampukan tidak terjadi pada keaadan biasa.Seperti dalam mod kawalan, hos perlu mengesahkan pemindahan data sahaja. Mod sampukan bertugas dengan hos yang berkaitan dengan peranti untuk melihat jika khidmat mod ini diperlukan.

Mod Bulk / Bulk mode and **Mod Isochronous / isochronous mode** berpadanan satu sama lain dalam satu keadaan. Mod Bulk / Bulk mode digunakan apabila data menepati kepentingan utama,tetapi kadar pemindahan data tidak boleh dijamin. Sebagai contoh. Storan Drive Disk. Mod Isochronous / Isochronous mode pula mengorbankan ketepatan data dalam jaminan pemasaan (timing) bagi penghantaran data. Contohnya ialah pembesar suara audio USB / USB audio speakers.

2.4 Mengapa Menggunakan USB?

USB dilaksanakan untuk menggantikan kewujudan port serial dan parallel pada komputer. Ia mempunyai pelbagai kebaikkan bagi aplikasi ini seperti :

- ***Penyambungan Peranti Yang Mudah*** - USB memudahkan penyambungan peranti luaran. USB menyokong " PLUG AND PLAY " iaitu operator tidak perlu terlibat terus dalam proses set-upnya kerana USB dapat mengenalpasti segera mana-mana peranti yang disambungkan pada hos bas USB.
- ***Bilangan Penyambungan Peranti Yang Banyak*** – Ia membenarkan bilangan peranti yang banyak dan besar bersambung dari satu hinggalah 127 peranti dari satu hos penyambung USB.
- ***Kadar Pemindahan Yang Tinggi*** - USB menggunakan kadar pemindahan data yang tinggi dari format data serial yang biasa.
- ***Compatible with any other device*** – USB merupakan satu peranti yang boleh digunakan dengan mana-mana sistem komputer dan peranti untuk berkomunikasi antara satu sama lain.
- ***Widely published, open standard*** – USB boleh dipunyai oleh mana-mana organisasi. Oleh kerana ia adalah open published, sesiapa sahaja boleh meningkatkan protocol penghantaran dan penerimaan data.

- ***Reliable, effient data delivery*** – USB boleh memastikan data di hantar dan diterima dengan cepat dan tepat
- ***Works on different hardware and network configurations*** – USB boleh diterima dan boleh dikonfigurasi untuk sebarang rangkaian yang direka.

Penerangan seterusnya akan dibincangkan dalam bab lima iaitu Bab Rekabantuk Sistem.

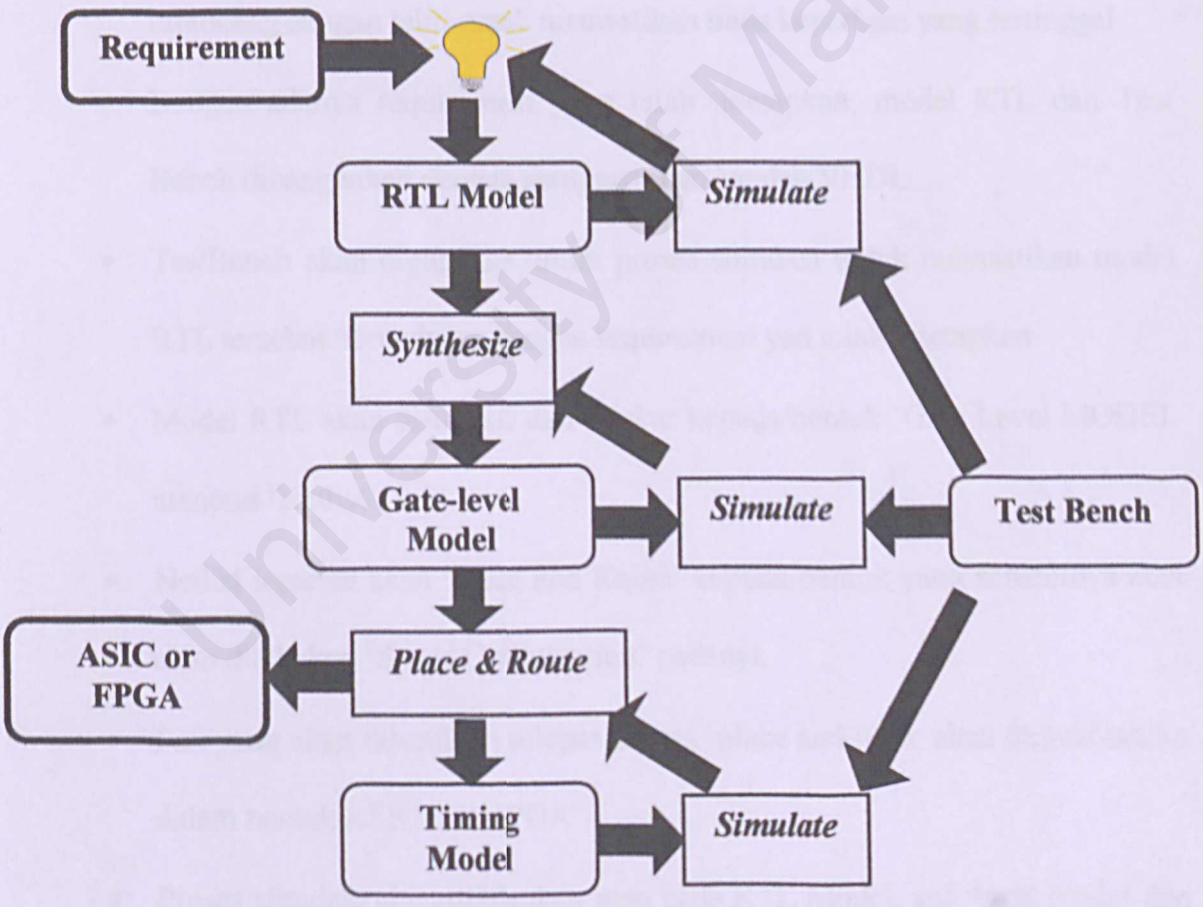
BAB TIGA

METODOLOGI

BAB 3 : SISTEM METODOLOGI VHDL

Bab ini membincangkan tentang metodologi projek ini. Metodologi merupakan satu set paduan yang lengkap di mana ia mengandungi model-model, kemudahan peralatan dan teknik-teknik khusus yang mesti diikuti dalam melaksanakan setiap aktiviti semasa membangunkan sistem.

3.1 Rekabentuk Asas Metodologi



Rajah 3.1

Metodologi ini digunakan kerana kaedah VHDL adalah lebih mudah untuk difahami dan diikuti daripada kaedah 'High Description Language' seperti Verilog. VHDL membawa maksud Very High Speed Integrated Circuit Hardware Description Language.

Gambarajah 3.1 di atas menunjukkan asas kepada kaedah rekabentuk mengikut konsep VHDL. Proses bermula dengan rekabentuk awalan hingga ke rekabentuk ASIC atau FPGA. Proses-proses tersebut adalah :

- Requirement / keperluan untuk setiap rekabentuk perlu ditentukan dan dirancang dengan teliti untuk memastikan tiada keperluan yang tertinggal
- Dengan adanya requirement yang telah ditetapkan, model RTL dan Test Bench dibangunkan dengan menggunakan kaedah VHDL
- TestBench akan digunakan untuk proses simulasi untuk memastikan model RTL tersebut berpadanan dengan requirement yan telah ditetapkan
- Model RTL akan disintesis dan ditukar kepada bentuk 'Gate Level MODEL ataupun 'Netlist'
- Netlist tersebut akan 'Place and Route' kepada bentuk yang sepatutnya atau akan dilakukan 'Speed Optimization' padanya
- Fail yang akan dihasilkan selepas proses 'place and route akan dimuatkan ke dalam bentuk ASIC atau FPGA
- Proses simulasi akan dijalankan juga pada RTL Model, get level model dan timing model untuk memastikan kesemuanya dapat berfungsi dengan betul.

BAB EMPAT

ANALISIS SISTEM

BAB 4: ANALISIS SISTEM

4.1 Sejarah VHDL

VHDL telah dibahagikan pada awal tahun 80-an sebagai projek penyelidikan integrasi litar yang berkelajuan tinggi. Ketika program VHSIC dijalankan, penyelidikan telah didedahkan dengan kebimbangan bagi menghuraikan litar yang berskala terlalu besar dan untuk menguruskan sebarang masalah rekabentuk litar yang bersaiz besar. Ketika itu hanya alatan rekabentuk aras get sahaja yang wujud, jadi penyelidik bersetuju dengan hakikat bahawa kaedah rekabentuk berstruktur perlu dihasilkan bagi mengatasi masalah yang sedia ada. Tiga syarikat iaitu IBM, Texas Instruments dan Inter Metrics telah memeterai kontrak dengan Department of Defense (DoD) untuk menyudahkan spesifikasi dan perlaksanaan bagi satu bahasa baru yang merupakan kaedah diskripsi bahasa rekabentuk. Versi pertama bagi bahasa tersebut adalah VHDL, versi 7.2 telah diwujudkan pada tahun 1985.

4.2 LatarBelakang

VHDL (Very High Speed Integrated Circuit Hardware Description Language) menjadi piawai IEEE 1076 dalam tahun 1987. Ia telah dikemaskini pada tahun 1993 dan kini dikenali sebagai 'IEEE Standard 1076 1993'. Verilog Hardware Description Language telah digunakan lebih lama daripada VHDL dan digunakan secara meluas sejak ia dilancarkan oleh Gateway pada tahun 1983. Cadence telah membeli Gateway pada tahun 1989 dan membuka perisian Verilog kepada umum pada tahun 1990. Ia menjadi piawai IEEE 1364 pada Disember 1995.

4.3 Apa itu VHDL

VHDL menggabungkan ciri-ciri berikut:

- Bahasa Pemodelan Simulasi

➢ Mempunyai pelbagai ciri-ciri yang sesuai untuk memerihalkan kelakuan bagi komponen elektronik bermula dengan get-get logik yang mudah kepada mikropemroses yang lengkap dengan cip-cip. Ciri-ciri yang ada pada VHDL membenarkan aspek elektronik bagi kelakuan litar seperti kenaikan dan kejatuhan masa bagi isyarat, kelambatan menerusi get an operasi kefungsian mudah diperihalkan.

- Kemasukan bahasa rekabentuk

➢ VHDL membenarkan kelakuan litar bagi litar elektronik yang kompleks diletakkan ke dalam sistem rekabentuk bagi sintesis litar automatik seperti Pascal, C dan C++. VHDL menggandingkan ciri-ciri yang berguna untuk teknik merekabentuk dan menawarkan set kawalan serta perwakilan data. VHDL membenarkan kejadian serentak (concurrent) dijelaskan.

- Bahasa Pengujian

➢ Ia berkeupayaan untuk mengetahui spesifikasi prestasi bagi 1 litar yang sering dirujuk sebagai testbench. Testbench merupakan perwakilan VHDL bagi stimulus litar dan persamaan output yang dijangka menunjukkan kelakuan bagi litar melepasi masa.

- Bahasa Netlist

➢ VHDL merupakan bahasa yang kuat sebagai kemasukan rekabentuk baru pada level yang tinggi malah ia juga berguna sebagai salah satu bahagian bagi level yang rendah sebagai komunikasi di antara alatan-alatan yang berbeza di dalam rekabentuk berpandukan komputer. Ciri bahasa berstruktur bagi VHDL membolehkan ia digunakan secara efektif sebagai bahasa netlist menggantikan bahasa netlist yang lain seperti EDIF.

4.4 Kelebihan VHDL

VHDL mempunyai rekabentuk digital seperti yang berikut :

- **Piawaian (Standard)** : VHDL sama seperti piwaian lain, ia mengurangkan kekeliruan dan menjadikan antaramuka di antara peralatan dan produk dengan mudah. Sebarang perkembangan kepada piawai mempunyai peluang yang lebih untuk bertahan lebih lama dan mengurangkan kekangan berbanding yang lain.
- **Meningkatkan produktiviti** : VHDL dapat meningkatkan produktiviti dengan memendekkan masa dalam pasaran. Simulasi boleh mengurangkan masa merekabentuk dengan membenarkan masalah rekabentuk di kesan dengan lebih awal bagi mengelakkan tugas rekabentuk pada peringkat get.
- **Sokongan industri** : Dengan adanya peralatan VHDL yang efisien dapat menyokong perkembangan di dalam industri elektronik.

- **Kebolehgunaan** : Terdapat sesetengah rekabentuk yang boleh diterangkan, dikenalpasti dan diubahsuai di dalam VHDL untuk kegunaan pada masa hadapan.
- **Mudah dibawa** : Kod VHDL yang sama boleh disimulasi dan digunakan dalam pelbagai alatan rekabentuk pada yang berbeza dalam proses rekabentuk. Ianya bergantung kepada set alatan rekabentuk yang mana kebolehannya adalah terhad dan tidak kompetitif dalam pasaran selepas ini. Piawaian VHDL juga menukarkan data rekabentuk dengan lebih mudah berbanding pengkalan data rekabentuk alatan yang menjadi hakmilik.
- **Kebolehan Secara Model** : VHDL dibangunkan kepada semua peringkat rekabentuk, iaitu daripada kotak elektronik kepada transistor. VHDL dapat menampung bentuk secara pelakuan dan rutin matematik yang menjelaskan tentang model yang kompleks seperti litar analog dan rangkaian. Ia juga membenarkan penggunaan senibina yang pelbagai dan bergabung dengan rekabentuk yang sama semasa pelbagai peringkat dalam proses rekabentuk.
- **Dokumentasi** : VHDL merupakan bahasa secara rekabnetuk yang membenarkan dokumentasi untuk diletakkan di dalam satu tempat denagn membenamkan (embedded) ia ke dalam bentuk kod. Gabungan antara komen dan kod ini sebenarnya adalah arahan yang patut dilakukan oleh rekabentuk bagi mengurangkan keraguan di antara penentuan dan pelaksanaan.

- **Metodologi Rekabentuk Baru** : Menggunakan VHDL dan sintesis untuk mencipta metodologi baru yang dapat meningkatkan produktiviti dalam rekabentuk, mengurangkan kitaran rekabentuk dan mengurangkan kos.

BAB LIMA

REKABENTUK SISTEM

BAB 5: REKABENTUK SISTEM

5.1 PENGENALAN

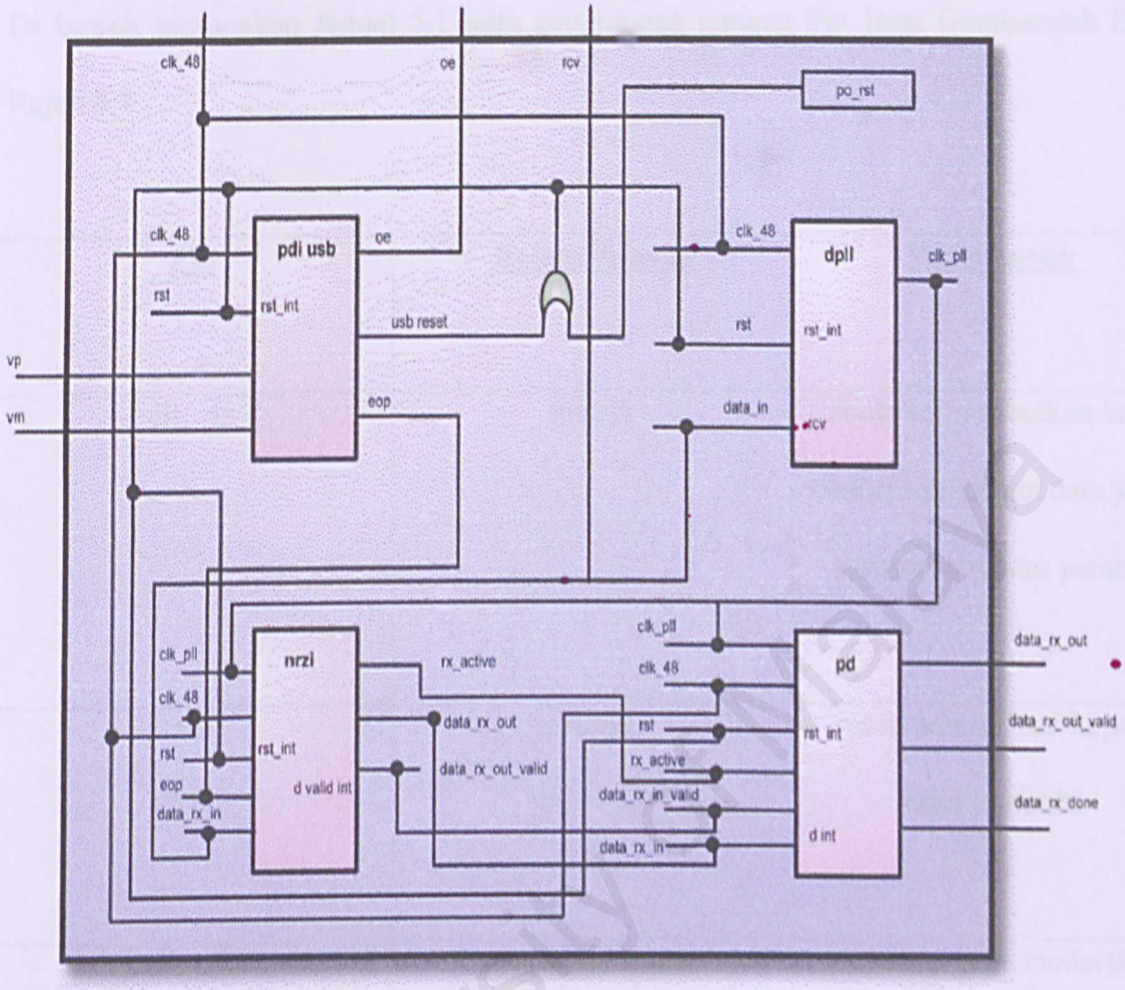
Seperti yang kita maklum, USB Transceiver merupakan peranti yang bertindak sebagai penghantar dan penerima data. Ianya berlaku apabila peranti USB dipasang pada port USB sistem komputer.

Telah dibincangkan bahawa USB menggunakan 4 kabel wayar antaramuka. Dua wayar tersebut digunakan dalam mod yang berbeza bagi kedua-dua penghantaran dan penerimaan data, manakala yang dua wayar lagi adalah untuk kuasa dan bumi. Sumber kuasa ke peranti USB boleh dating dari hos, hub atau peranti *"self powered"*.

Terdapat dua jenis penyambung yang berlainan pada setiap hujung kabel USB. Salah satunya adalah untuk komunikasi *upstream* dan yang satu lagi adalah untuk *downstream.* Setiap panjang kabel adalah terhad kapeda 5 meter.

USB mempunyai empat jenis mod pemindahan komunikasi :

- Kawalan / control,
- Sampukan / interrupt,
- bulk, dan
- isochronous.



Rajah 5.1

Di bawah merupakan Jadual 5.1 iaitu penerangan tentang Pin Bagi Gambarajah Blok Rajah 5.1

Pin	**Keluar/Masuk**	**Penerangan**
clk_48	masuk	'clock' ini mensetkan kadar maksimum setiap data yang masuk ke dalam peranti
rst	masuk	'reset' semua litar kepada nilai 'default'
clk_pll	masuk	clock' ini juga mensetkan kadar maksimum setiap data yang masuk ke dalam peranti dpll
eop	masuk	paket yang terakhir masuk dan akan menghentikan paket pengeluaran data

rx_active	masuk	menunjukkan penerimaan data yang aktif. Ianya akan dihentikan apabila eop dikesan
data_in	masuk	data masuk
data_rx_in	masuk	data diterima masuk
data_rx_in_valid	masuk	data dikenalpasti dan diterima masuk
Vp	masuk	isyarat masuk
Vm	masuk	isyarat masuk
oe	masuk	Membenarkan data yang keluar masuk ke destinasi yang seterusnya
rcv	masuk	menerima data
usb reset	keluar	mengaktifkan kembali usb

eop	keluar	data keluar
rx_active	keluar	menunjukkan penerimaan data yang aktif. Ianya akan dihentikan apabila eop dikesan
rx_active_out	keluar	data aktif keluar
data_rx_out	keluar	data keluar
data_rx_out_valid	keluar	data yang dikenalpasti keluar
data_rx_done	keluar	data yang selesai dan diterima

Jadual 5.1

5.2 Cylic Redundency Check (CRC)

CRC merupakan kod pengesanan ralat yang sangat popular dalam skim penghantaran data. Ia sebagai penyemak lewahan berkitar iaitu satu bentuk pengiraan matematik ke atas blok data dan akan kembalikan kandungan data tersebut bagi mengetahui maklumat yang dihantar benar atau salah.

CRC ini berasaskan bit yang ditunjukkan oleh polinomial. Contohnya ialah 110001 (mempunyai 6 bit), maka penjanaan polinomial ditunjukkan oleh: G(X)=X5 + X4 + 1. G(X) ini bermaksud penjana polinonial.

5.2.1Contoh Pengiraan CRC

Pesan = 110101

Polinomial = 101

11010100 / 101

11010100

101

111

101

100

101

110

101

11 baki = CRC checksum

5.2.2 Dasar Teori CRC

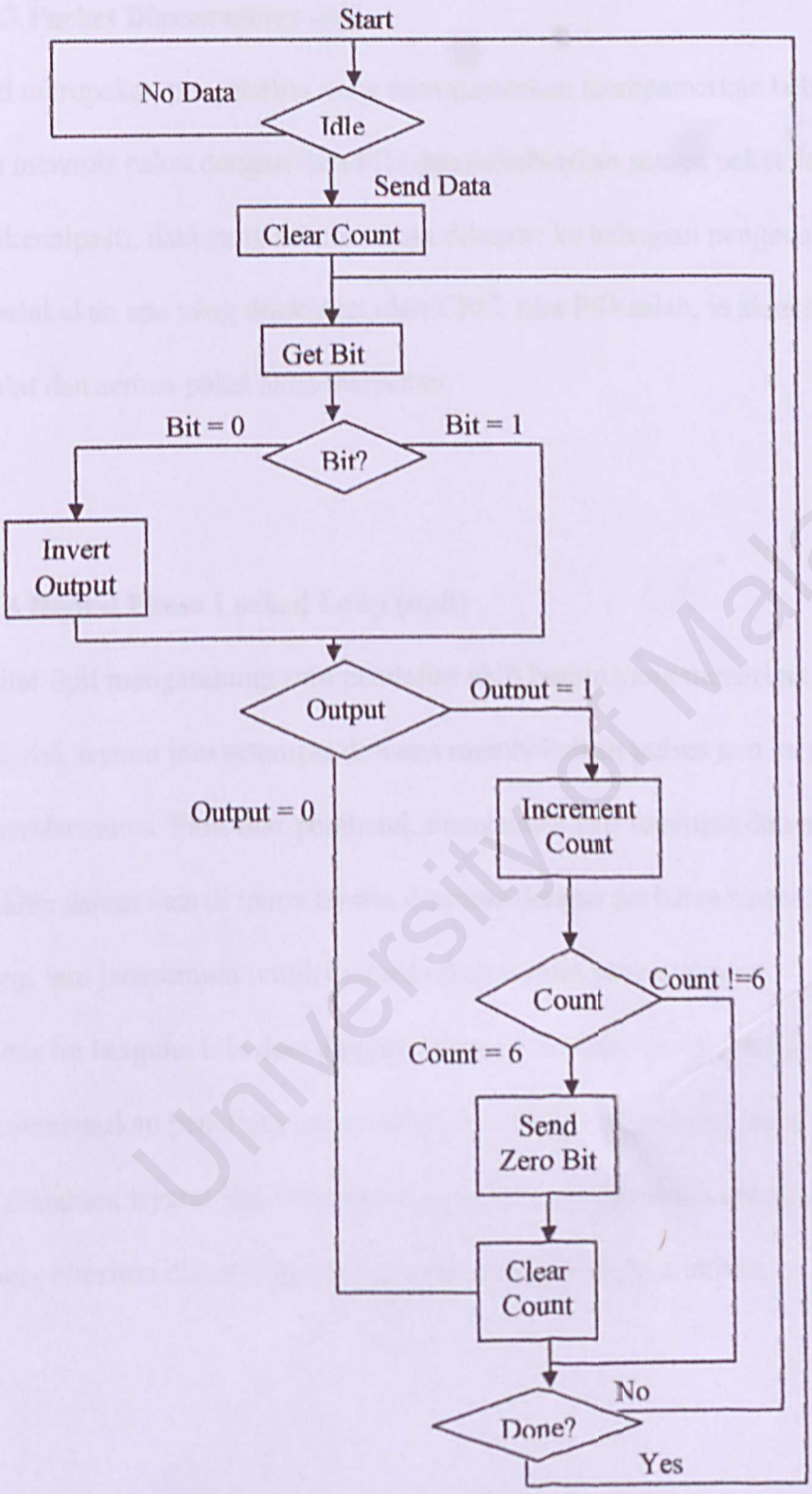
CRC Merupakan kod pemeriksaan ralat yang umum digunakan pada sistem komunikasi data dan sistem transmisi data bersiri yang lain. Selain itu, terdapat pelbagai piawai CRC yang lain seperti :

- CRC – 8
- CRC – 16
- CRC – 32
- CRC – CCIT

Algoritma CRC – 16 biasanya digunakan jika panjang pesan mencapai 1kb atau lebih sehingga pesan harus ditransmisikan antara unit yang terpisah. Jika panjang pesan (message) mencapai beberapa kilobait dan harus ditransmisikan pada saluran yang kurang baik seperti radio link, maka algoritma yang digunakan adalah CRC-32.

Di mukasurat sebelah merupakan carta alir yangterlibat dalam proses CRC dan bit stuffing.

Rajah 5.2 : Carta Alir yang digunakan untuk pergerakan data USB bagi CRC dalam proses Bit Stuffing



5.3 Packet Disassembler (pd)

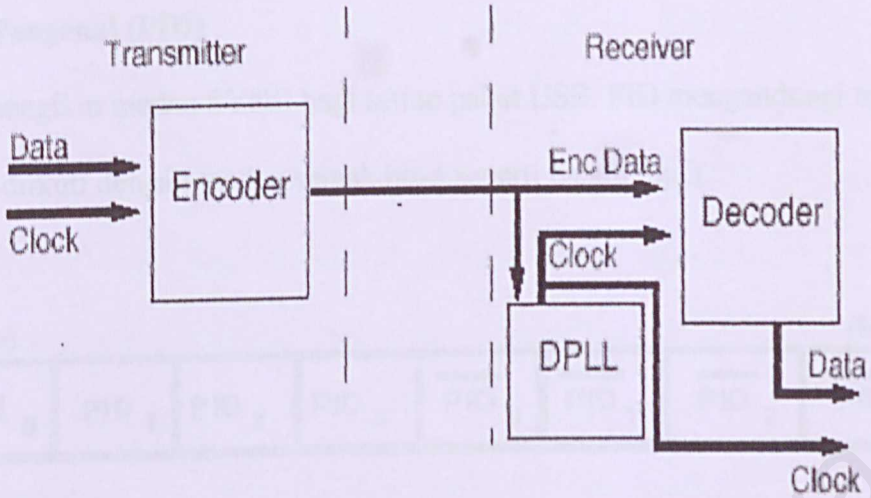
Pd merupakan penganalisa yang mempamerkan mempamerkan beberapa fungsi.

Ia menapis paket dengan data PID dan membiarkan semua paket yang lain. Selepas data dikenalpasti, data yang sebenar akan dihantar ke bahagian pengenal imej. Ia tidak melakukan apa yang dilakukan oleh CRC. Jika PID salah, ia akan menjanakan keadaan ralat dan semua paket akan diabaikan.

5.4 Digital Phase Locked Loop (dpll)

Litar dpll mengandungi satu pendaftar shift bersiri yang menerima sampel kemasukkan digital, isyarat jam setempat di mana membekalkan pulses jam ke pendaftar shift utk membawanya. Fasa litar pembetul, mengambil jam setempat dan menjanasemula jam stable dalam fasa di mana isyarat diterima dengan perlahan membetulkan fasa untuk fasa bagi jam janasemula untuk memadankan isyarat yang diterima.

Litar ini berguna bila data dan jam dihantar bersama melalui kabel biasa. Sejak ia membenarkan penerima mengasingkan (janasemula) isyarat jam dari data yang diterima. Kemudian, isyarat jam yang dijanasemula akan digunakan untuk mensampelkan data yang diterima dan mengenalpasti nilai setiap bit yang diterima.



Rajah 5.3

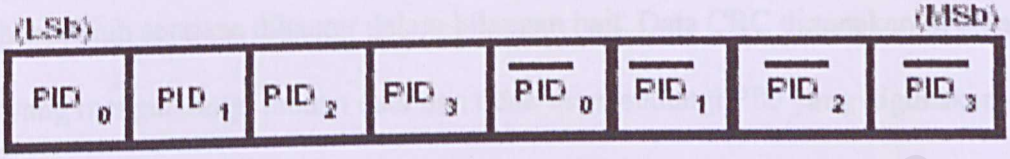
5.5 Format Paket

Format paket ini menjana kepadatan transisi bucu maksimum (the maximum of the edge transition). Medan SYNC muncul pada bas sebg 'IDLE' diikuti dgn aksara binari "0101 0100", dalam pengkodan NRZI.

Ia digunakan dgn litar input utk samakan data masuk dgn 'local clock' & jadikan panjangnya 8 bit. SYNC hanya untuk pengsinkronas. 2 bit akhir dalam medan merupakan penanda yg digunakan utk mengenalpasti hujung medan SYNC & mulanya PID.

5.6 Paket Pengenal (PID)

Ia segera mengikut medan SYNC bagi setiap paket USB. PID mengandungi medan jenis paket bit-4 diikuti dengan medan semak bit-4 seperti dalam rajah.

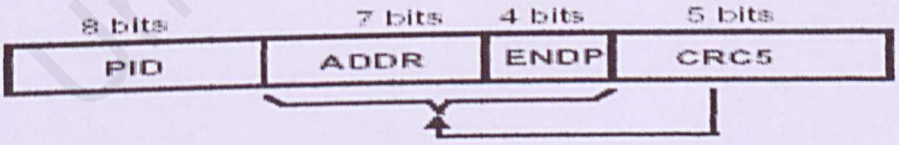


Rajah 5.4

5.6.1 Jenis Paket Dalam USB

Paket – paket di dalam USB ini mempunyai pelbagai jenis. Ada tiga jenis paket iaitu :

- Token – Digunakan utk alamat (address).
- Data - Digunakan utk data.
- Handshake – Digunakan utk hentikan perubahan data.

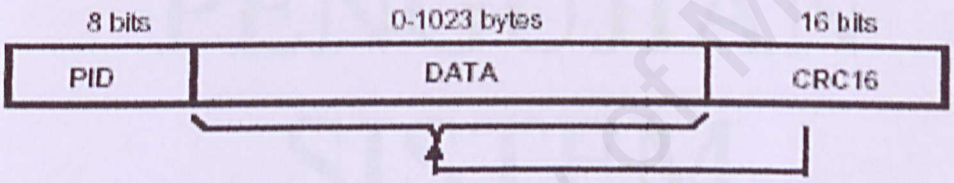


Rajah 5.5

5.7 Format Paket Data

Paket data mengandungi PID, medan data mengandungi nilai kosong atau lebih bait data dan CRC seperti dalam Gambarajah Terdapat 2 jenis paket data yang dikenalpasti oleh 2 PID yang berbeza iaitu DATA 0 dan DATA 1.Dua paket data PID digunakan untuk menyokong data 'toggle' bagi keadaan penambahan secara segerak.

Data hendaklah sentiasa dihantar dalam bilangan bait. Data CRC digunakan di dalam paket yang mengandungi medan data dan tidak mengandungi PID yang digunakan untuk mengesan medan tersebut. Ia dapat dilihat seperti di dalam Rajah 5.6.



Rajah 5.6

BAB ENAM

PENGUJIAN SISTEM

6.1 PENGENALAN

Di dalam bab ini, saya akan menerangkan tentang bagaimana sesuatu modul itu apabila dibangunkan serta kaedah dan langkah-langkah yang diambil untuk menguji keberkesanan sesuatu modul seperti yang dikehendaki.

Pengujian merupakan langkah yang diambil bagi memastikan sesuatu modul itu dapat dilaksanakan mengikut arahan-arahan yang telah ditetapkan oleh saya sendiri.

Dengan melakukan pengujian ke atas modul yang telah dibangunkan, dapat saya mengesan sebarang ralat yang berlaku pada modul iaitu ke atas kod-kod yang telah ditulis. Selain daripada mengesan ralat yang berlaku, teknik pengujian dapat memastikan tidak berlakunya sebarang kesalahan pada kod-kod yang telah ditulis dan memastikan ianya dapat beroperasi seperti yang dikehendaki.

6.2 PEMBANGUNAN MODUL

Bagi memastikan sesuatu modul itu dapat berfungsi dengan lancar, saya telah memilih untuk menghasilkan sesuatu modul besar kepada modul-modul yang kecil bagi memudahkan pengesanan ke atas ralat yang berlaku. Langkah ini bertujuan untuk mempercepatkan pengesanan ke atas ralat yang berlaku ke atas kod-kod yang telah ditulis. Sebagai contoh, saya telah membahagikan satu modul unit cyclic redundancy checker (CRC) kepada encoder dan decoder.

6.3 PEMBANGUNAN TEST BENCH

Modul test bench dibangunkan secara serentak dengan modul utama. Test bench bertujuan untuk menguji kebolehlaksanaan sesuatu modul yang telah dibangunkan. Dengan membangunkan testbench, saya dapat mengetahui sesuatu modul telah berfungsi seperti yang dikehendaki apabila ianya disimulasikan kemudiannya. Bagi aturan di atas, contoh test bench yang dihasilkan adalah seperti berikut.

6.4 PENGUJIAN DENGAN PERISIAN PEAK FPGA

Perisian PEAK FPGA telah digunakan untuk membangunkan projek ini. Perisian ini menyediakan servis yang mudah bagi kegunaan pengguna. Saya akan menerangkan tentang kemudahan ataupun ciri-ciri yang disediakan dalam membangunkan modul. Tiga menu utama yang penting dalam pengujian menggunakan perisian ini adalah pengkompil (compile), pautan (link) dan simulasi (simulation).

6.5 PENGKOMPIL

Perisian ini menyediakan khidmat kompil untuk memudahkan saya mengetahui jika ada ralat pada modul yang ditulis oleh saya dengan mengeluarkan dialog pemberitahuan ralat. Contoh ralat yang selalu berlaku adalah ralat dari segi ketidaksamaan saiz bit pada pemetaan port dengan sub modul.

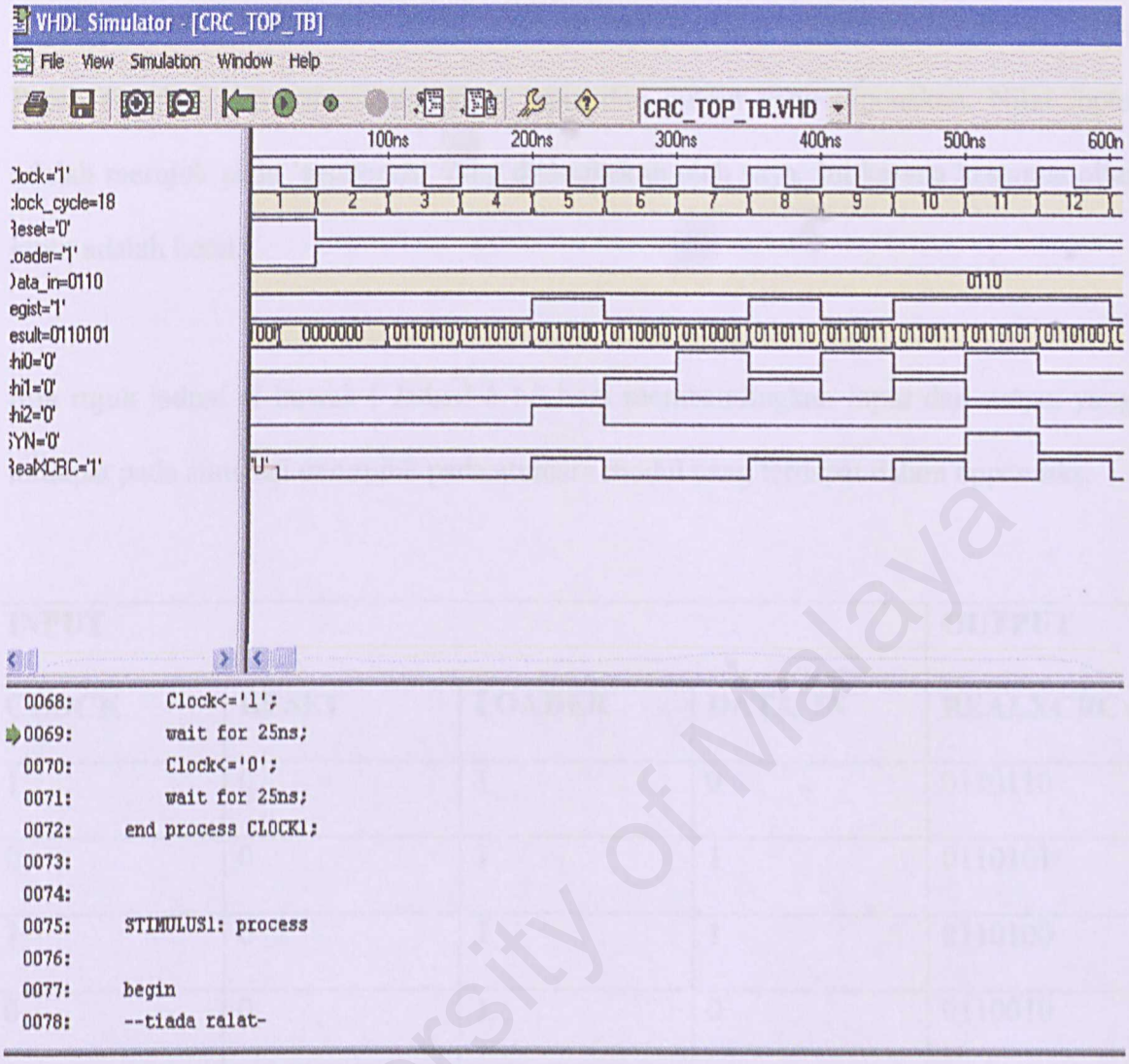
6.6 PAUTAN (LINK) DAN PEMETAAN PORT (PORT MAP)

Link digunakan untuk membolehkan sesuatu modul itu diintegrasi dengan modul lain. Sebelum itu, ia berfungsi untuk menghubungkan modul dengan testbench.

Port map merupakan satu kaedah di mana setiap modul yang dibangunkan tadi digabungkan dengan menyediakan satu lagi modul port map yang tersendiri. Ia melibatkan semua sub modul dan modul utama (top level).

6.7 SIMULASI

Ia merupakan antara yang paling penting dalam memastikan sesuatu modul itu berfungsi seperti yang dikehendaki (expected output). Simulasi menunjukkan aliran input, output atau isyarat (signal) yang diisytiharkan melalui bentuk gelombang atau 'waveform' . daripada simulasi itu, kita boleh periksa input yang dimaksudkan dan output yang terhasil seperti yang dijangkakan.



Rajah 6.1 : Simulasi bagi modul utama CRC (CRC Top Level)

Rajah simulasi (Rajah 6.1) di halaman sebelah merujuk kepada modul utama CRC. Rajah simulasi tersebut menunjukkan input dan output yang digunakan. Nilai input adalah merujuk pada 'testbench' yang dijangkakan oleh saya. Ini kerana kesemua nilai input adalah betul.

Sila rujuk jadual di bawah (Jadual 6.1) bagi membandingkan input dan output yang terdapat pada simulasi dan rujuk pada aturcara modul yang terdapat dalam appendiks.

INPUT				**OUTPUT**
CLOCK	**RESET**	**LOADER**	**DATA_IN**	**REALXCRC**
1	0	1	0	0110110
0	0	1	1	0110101
1	0	1	1	0110100
0	0	1	0	0110010

Jadual 6.1 : Pernyataan input dan output bagi Modul Utama CRC

BAB TUJUH

PERBINCANGAN DAN KESIMPULAN

7.1 PENGENALAN

Secara keseluruhan, saya kurang berjaya dalam mencapai hasil output yang dikehendaki terutama bagi modul DPLL dan PD. Ini kerana terdapat beberapa kekangan yang dihadapi semasa membangunkan perisian ini, terdapat modul komponen seperti di atas tidak dapat dihasilkan semasa membangunkan rekabentuk ini.

Dalam bab ini, ingin saya membincangkan beberapa masalah atau kekangan yang dihadapi oleh saya semasa membangunkan projek ini. Antaranya ialah, perubahan-perubahan yang dilakukan terhadap rekabentuk, masalah-masalah yang dihadapi beserta penyelesaian, cadangan di masa hadapan dan juga kesimpulan terhadap projek yang dijalankan sepanjang semester dua ini.

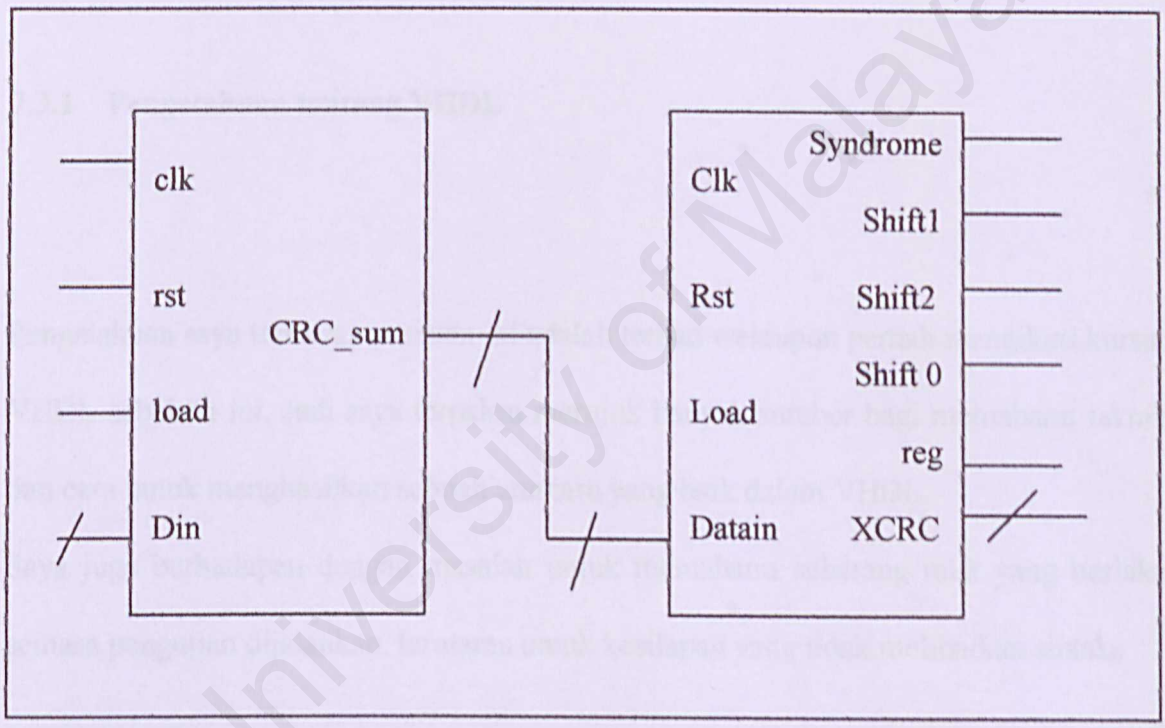
7.2 PERUBAHAN TERHADAP REKABENTUK.

Sekiranya diperhatikan pada rekabentuk modul yang dipaparkan dalam bab 5 iaitu rekabentuk yang dicadangkan, terdapat perubahan yang telah dilakukan bagi memenuhi skop projek. Antara perubahan yang dilakukan adalah :

7.2.1 Rekabentuk terbaru menambah satu modul CRC

Penambahan ini adalah untuk melengkapkan fungsi komponen di dalam USB. Ia mempunyai kaitan secara menyeluruh terhadap semua gabungan modul di dalam USB seperti dpll, pd, pdi, nrzi dan juga bit-stuffed.

Di bawah adalah rajah blok diagram bagi CRC.



7.3 MASALAH PEMBANGUNAN YANG DIHADAPI

Di dalam pembangunan projek ini, tentunya saya tidak dapat mengelak dari masalah bagi membangunkan testbench untuk dpll dan pd. Maka itu, saya dengan segera mencari jalan penyelesaian dengan segera supaya masalah ini tidak wujud pada masa akan dating dalam membangunkan projek ini. Antara masalah – masalah yang dihadapi oleh saya :

7.3.1 Pengetahuan tentang VHDL

Pengetahuan saya terhadap perisian ini adalah terhad walaupun pernah mengikuti kursus VHDL sebelum ini. Jadi saya terpaksa merujuk banyak sumber bagi memahami teknik dan cara untuk menghasilkan sebuah aturcara yang baik dalam VHDL.

Saya juga berhadapan dengan masalah untuk memahami sebarang ralat yang berlaku semasa pengujian dijalankan, terutama untuk kesilapan yang tidak melibatkan sintaks

dan jarang dijumpai oleh saya seperti "aborting compile" , amaran – amaran semasa mengkompil dan "multiresources".

Penyelesaian

Saya melakukan kajian dan pembelajaran yang lebih terhadap perisian ini daripada sumber di internet dan buku rujukan bagi memahirkan diri dengan VHDL. Hasilnya, walaupun saya kurang mahir dalam perisian ini iaitu penggunaan bahasa pengaturcaraan VHDL, saya tetap dapat menambahkan lagi pengetahuan dari sebelumnya. Selain itu, penyelia projek dan rakan-rakan banyak membantu dalam projek ini.

7.3.2 Had Masa

Untuk menghasilkan satu simulasi untuk komponen-komponen USB, pastinya memakan masa yang panjang. Jadi dengan tempoh yang terhad sepanjang semester kedua ini, saya hanya sempat menghasilkan modul CRC sahaja.

Penyelesaian

Saya harus lebih peka dan pandai dalam membahagikan masa terhadap pembangunan projek ini. Pembahagian masa yang lebih efisyen harus dibentuk untuk membolehkan saya memberi sepenuh perhatian dan tumpuan terhadap pembangunan projek ini. Pembahagian masa adalah penting dalam membolehkan saya menyiapkan projek ini pada masa yang ditetapkan.

7.3.3 Sumber Rujukan

Rujukan merupakan satu sumber yang penting di dalam membuat kajian dan membangunkan projek ini. Tapi saya mempunyai masalah bahan rujukan yang kurang. Contohnya, perpustakaan mempunyai kurang sumber rujukan tentang perisian bagi bahasa pengaturcaraan VHDL. Buku yang sedia ada adalah sudah lama dan disediakan dalam kuantiti yang keci dan terhad.

Penyelesaian

Memandangkan sumber rujukan adalah terhad dan kurang, saya telah mengabil inisiatif dengan mencari maklumat terkini tentang USB dan perisian di pelbagai enjin pencarian (search engine) internet. Diharap agar pihak perpustakaan dapat menambahkan lagi sumber rujukan dan edisi terkini berkaitan dengan VHDL.

7.3.4 Perkakasan dan perisian

Masalah yang dihadapi adalah berkaitan dengan perkakasan yang dimiliki oleh saya tidak menepati spesifikasi yang diperlukan oleh PEAK FPGA yang memerlukan komputer berkuasa tinggi bagi membolehkan ia dilarikan dengan lancer. Penggunaan

komputer adalah terhad di makmal sahaja iaitu pada waktu dan petang. Jadi, ia sedikit sebanyak mengganggu usaha saya dalam membangunkan projek ini.

Penyelesaian

Saya telah membuat draf bagi setiap modul yang dibangunkan. Ini bagi membolehkan saya menggunakan kemudahan di makmal dengan semaksima yang mungkin utnuk menyiapkan modul-modul tersebut.

7.4 CADANGAN MASA HADAPAN

Sekiranya, projek ini diteruskan pada masa hadapan, saya dapati terdapat ciri-ciri yang boleh ditambah dan diperbaiki untuk membolehkan rekabentuk perkakasan ini menepati apa yang dirancangkan. Ciri-ciri tersebut adalah seperti berikut :

7.4.1 Penambahan modul untuk multiplexer

Dalam menghasilkan modul encode dan decode adalah sesuai jika ditambah dengan modul multiplexer kerana fungsinya dapat dijadikan satu cara dalam membuat pilihan samada hendak menggunakan encode atau decode.

7.4.2 Menyiapkan testbench untuk semua modul yang ada

Sekiranya saya mempunyai peluang dan masa yang lebih panjang ingin saya menyiapkan semua testbench bagi setiap modul yang ada seperti dpll dan pd. Jika semua test bench dapat disiapkan, maka sempurnalah projek saya dan satu diagram USB dapat ditunjukkan dari segi fungsi bagaimana USB menghantar dan menerima data.

7.5 KESIMPULAN

Alhamdulillah, walaupun projek ini tidak dapat disiapkan mengikut perancangan, namun ia tetap mencapai objektifnya, walaupun saya hanya merekabentuk modul semakan ralat iaitu CRC, namun idea pembangunannya boleh dikembangkan lagi untuk menghasilkan modul-modul yang lain.

Pembangunan kod sumber bagi perkakasan merupakan suatu perkara yang kompleks berbanding pembangunan kod sumber untuk perisian. Segala perancangan tahap awal agak mudah tetapi berbeza pula untuk tahap akhir iaitu dari segi perlaksanaan dan pengujian sistem.

Walaupun kod sumber telah dihasilkan betul dan dapat menghasilkan keluaran yang sepatutnya, namun litar dalam perkakasan boleh memberi kesan yang sebaliknya kepada data output yang dihasilkan.

Secara keseluruhan, projek ini telah memberikan saya satu ilmu yang baru serta pengalaman yang amat berguna dalam merekabentuk satu perkakasan. Semasa projek ini juga saya telah melalui kaedah merekabentuk seperti fasa pembangunan, pengkodan dan pengujian. Saya juga mempelajari tentang ciri-ciri dan kelakuan perkakasan, iaitu sesuatu yang tidak pernah saya ketahui sebelum ini.

APPENDIKS A

KODING DAN TESTBENCH

```
-----CODING UNTUK TOP LEVEL CRC----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    port ( Clock: in std_ulogic;
           Reset: in std_ulogic;
          Loader: in std_ulogic;
          Data_in: in std_ulogic_vector(3 downto 0);
          Result: out std_ulogic_vector(6 downto 0);
         syn: out std_ulogic;
          shi2: out std_ulogic;
          shi1: out std_ulogic;
          shi0: out std_ulogic;
          regist: out std_ulogic;
          RealXCRC: out std_ulogic);

end top;

architecture structural of top is


--component

component CRC_ENCODE

port (  clk: in std_ulogic;
        rst: in std_ulogic;
        load: in std_ulogic;
        Din: in std_ulogic_vector(3 downto 0);
        CRC_sum: out std_ulogic_vector(6 downto 0));
end component;

component CRC_DECODE

port (  clk: in std_ulogic;
        rst: in std_ulogic;
        load: in std_ulogic;
        Datain: in std_ulogic_vector(6 downto 0);
        syndrome: out std_ulogic;
        shift2: out std_ulogic;
        shift1: out std_ulogic;
        shift0: out std_ulogic;
        reg: out std_ulogic;
        XCRC: out std_ulogic);
end component;

--signals
signal outsig : std_ulogic_vector(6 downto 0);
signal sind : std_ulogic;
signal s2 : std_ulogic;
signal s1 : std_ulogic;
signal s0 : std_ulogic;
signal re : std_ulogic;
```

```
signal xxx : std_ulogic;

begin

AAA: CRC_ENCODE
      port map(clk=>Clock,
               rst=>Reset,
               load=>Loader,
               Din=>Data_in,
               CRC_sum=>outsig);
BBB: CRC_DECODE
      port map(clk=>Clock,
               rst=>Reset,
               load=>Loader,
               Datain=>outsig,
               syndrome=>sind,
               shift2=>s2,
               shift1=>s1,
               shift0=>s0,
               reg=>re,
               XCRC=>xxx);

syn<=sind;
shi2<=s2;
shi1<=s1;
shi0<=s0;
regist<=re;
RealXCRC<=xxx;
Result<=outsig;
end structural;
```

```
--- CODING UNTUK MODUL CRC ENCODE----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity CRC_ENCODE is
    port (

        clk: in std_ulogic;
        rst: in std_ulogic;
        load: in std_ulogic;
        Din: in std_ulogic_vector(3 downto 0);
        CRC_sum: out std_ulogic_vector(6 downto 0)

    );

end CRC_ENCODE;

architecture BEHAVIOR of CRC_ENCODE is

signal X  : std_ulogic_vector(6 downto 0);
signal s_d: std_ulogic_vector(3 downto 0);

begin

   s_d<=Din;

process(clk,rst)

    begin
        if rst = '1' then
           X<=(others=>'0');
   CRC_sum<="0000000";

elsif rising_edge(clk) then

if(load='1') then
    s_d<='0'& s_d(3 downto 1);
    X(6 downto 3)<=(others=>'0');
     X(2)<=s_d(0)xor X(0);
     X(1)<=s_d(0)xor X(2)xor X(0);
     X(0)<=X(1);

end if;

end if;

end process;

CRC_sum<= Din & X(2 downto 0);

end BEHAVIOR;
```

```
--- CODING UNTUK CRC DECODE ----



library ieee;
use ieee.std_logic_1164.all;


entity CRC_DECODE is
    port (

        clk: in std_ulogic;
        rst: in std_ulogic;
        load: in std_ulogic;
        Datain: in std_ulogic_vector(6 downto 0);
        syndrome: out std_ulogic;
        shift2: out std_ulogic;
        shift1: out std_ulogic;
        shift0: out std_ulogic;
        reg: out std_ulogic;
        XCRC: out std_ulogic

    );

end CRC_DECODE;

architecture BEHAVIOR of CRC_DECODE is

signal X                :std_ulogic_vector(6 downto 0);
signal s_d                 :std_ulogic_vector(6 downto 0);
signal s_out            :std_ulogic;
signal syn, s_syn1,s_reg   :std_ulogic;



begin

   s_d<=Datain;

    process (clk, rst)

    begin
        if rst = '1' then

          X<=(others =>'0');
          reg <='0';
          XCRC<='0';

        elsif rising_edge(clk) then
s_d<='0' & s_d(6 downto 1);
X(6 downto 3)<=(others=>'0');
X(2)<=X(0)xor s_d(0);
X(1)<=X(2)xor X(0);
X(0)<=X(1);
```

```
reg<=s_d(0);
s_out<=syn xor s_d(0);
s_reg<=s_d(0);

end if;

--end if;

end process ;

syn<=(X(2)and (not X(1)))and X(0);
s_syn1<=(X(2) and (not X(1))) and X(0);
syndrome<=syn;
XCRC<=s_syn1 xor s_reg;
shift2<=X(2);
shift1<=X(1);
shift0<=X(0);



end BEHAVIOR;
```

```
---- CODING UNTUK TESTBENCH CRC----


library ieee;
use ieee.std_logic_1164.all;

use std.textio.all;
use work.top;

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component top is
    port (

        Clock: in std_ulogic;
        Reset: in std_ulogic;
        Loader: in std_ulogic;
        Data_in: in std_ulogic_vector(3 downto 0);
        Result: out std_ulogic_vector(6 downto 0);
      syn: out std_ulogic;
        shi2: out std_ulogic;
        shi1: out std_ulogic;
        shi0: out std_ulogic;
        regist: out std_ulogic;
        RealXCRC: out std_ulogic
    );

end component;
constant PERIOD: time :=50 ns;

--Clk: in std_logic;

signal Clock:  std_ulogic;
signal Reset: std_ulogic;
signal Loader: std_ulogic;
signal Data_in: std_ulogic_vector(3 downto 0);
signal Result:  std_ulogic_vector(6 downto 0);
signal syn: std_ulogic;
signal shi2: std_ulogic;
signal shi1: std_ulogic;
signal shi0: std_ulogic;
signal regist:  std_ulogic;
signal RealXCRC:  std_ulogic;

--signal done: boolean := false;
signal clock_cycle: natural :=0;


begin
    DUT: top port map (
        Clock,
        Reset,
        Loader,
```

```
        Data_in,
        Result,
       syn,
        shi2,
        shi1,
        shi0,
        regist,
        RealXCRC

);

    CLOCK1: process
         begin
      clock_cycle<=clock_cycle + 1;
       Clock<='1';
       wait for 25ns;
       Clock<='0';
       wait for 25ns;
    end process CLOCK1;


    STIMULUS1: process

    begin
    --tiada ralat-
    Loader<='0';
    Reset<='1';
    Data_in<="0111";
    wait for PERIOD;
    Loader<='1';
    Reset<='0';
    Data_in<="0111";
    wait for PERIOD;

     Loader<='1';
    Reset<='0';
    Data_in<="0110";
    wait for PERIOD;


    --ralat
    --load<='0';
    --rst<='1';
    --Din<="0111";
    --wait for PERIOD;
    --load<='1';
    --rst<='0';
    --Din<="0111";
    --wait for PERIOD;

     wait;
end process STIMULUS1;

end stimulus;
```

```
--- CODING UNTUK DPLL ---


library ieee;
use ieee.std_logic_1164.all;

entity dpll is

port ( data_in:  in std_logic;
   rst:          in std_logic;
   clk_48:          in std_logic;
   clk_pll:     out std_logic
   );

end dpll;

-------------------------------------------------------------------
-- Architecture of  dpll (digital phase locked loop)
-------------------------------------------------------------------

architecture dpll_arch of dpll is

type pll_state is (sc,sd,s5,s7,s6,s4,sf,sb,s1,s3,s2,s0);
signal state:pll_state;
signal a, b: std_logic;

begin

sync_a: process(clk_48, rst)
begin
   if rst='1' then
      a <= '0';
   elsif (clk_48'event and clk_48='1') then
      a <= data_in;
   end if;
end process;

sync_b: process(clk_48, rst)
begin
   if rst='1' then
      b <='1';
   elsif (clk_48'event and clk_48='0') then
      b <= data_in;
   end if;
end process;

clk_gen: process(clk_48, rst)
begin
   if rst ='1' then
      state <= sc;
      clk_pll<='0';
   elsif (clk_48'event and clk_48='1') then
      case state is
         when sc =>
            if b='0' then
               state <= sd;
```

```
            clk_pll<='0';
        else state <=sc;
            clk_pll<='0';
        end if;
    when sd =>
        if b='1' then
            state <= s5;
        else state <=sd;
        end if;
    when s5 =>
            state <= s7;
            clk_pll<='1';
    when s7 =>
        if a='1' then
            state <= s6;
        else
            state <= sb;
        end if;
    when s6 =>
        if b='1'  then
            state <= s4;
            clk_pll <='0';
        else state <= s1;
            clk_pll <='0';
        end if;
    when s4 =>
        if b='1' then
            state <= s5;
        elsif b='0' then
            state <= s1;
        end if;
    when s1 =>
            state <= s3;
            clk_pll<='1';
    when s3 =>
        if a='0' then
            state <= s2;
        else state <= sf;
        end if;
    when s2 =>
        if b='0' then
            state <= s0;
            clk_pll<='0';
        else state <= s5;
            clk_pll<='0';
        end if;
    when s0 =>
        if b='0' then
            state <= s1;
        elsif b='1' then
            state <= s5;
        end if;
    when sb =>
            state <= s2;
    when sf =>
            state <= s6;
end case;
```

```
   end if;
end process clk_gen;

end dpll_arch;



-- VHDL Test Bench Created from source file dpll.vhd -- 17:45:02
02/07/2005


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY dpll_ujidpll_vhd_tb IS
END dpll_ujidpll_vhd_tb;

ARCHITECTURE behavior OF dpll_ujidpll_vhd_tb IS

   COMPONENT dpll
   PORT(
      data_in : IN std_logic;
      rst : IN std_logic;
      clk_48 : IN std_logic;
      clk_pll : OUT std_logic
      );
   END COMPONENT;

   constant PERIOD: time:= 50ns;

   SIGNAL data_in :  std_logic;
   SIGNAL rst :  std_logic;
   SIGNAL clk_48 :  std_logic;
   SIGNAL clk_pll :  std_logic;

BEGIN

   dut: dpll PORT MAP(
      data_in => data_in,
      rst => rst,
      clk_48 => clk_48,
      clk_pll => clk_pll
   );

    clock1: PROCESS
    BEGIN
    clk_48<='0';
    wait for period/2;
    clk_48<='1';
    wait for period/2;
    END PROCESS;

-- *** Test Bench - User Defined Section ***
```

```
   tb : PROCESS

   BEGIN

--clock gen

   rst<='1';
   data_in<='1';
   wait for period;--Wait One clock cycle


   rst<='0';
   data_in<='1';

  -- wait for period*5;--Wait One clock cycle
 --  data_in<='0';
 --  rst<='0';
   wait for period;--Wait One clock cycle


   data_in<='0';
   rst<='1';
   wait for period;--Wait One clock cycle

--*** sc
   data_in<='1';
   rst<='0';
   wait for period;--Wait One clock cycle

--*** sd
   data_in<='0';
   rst<='0';
   wait for period;--Wait One clock cycle

 --*** s7
   data_in<='1';
   rst<='0';
   wait for period;--Wait One clock cycle


--*** s6
  --data_in<='0';
  -- rst<='1';
  --wait for period;--Wait One clock cycle

--*** s4
  --data_in<='0';
  -- rst<='1';
  --wait for period;--Wait One clock cycle

--*** s1
 -- data_in<='0';
  --rst<='0';
  --wait for period;--Wait One clock cycle


--*** s
```

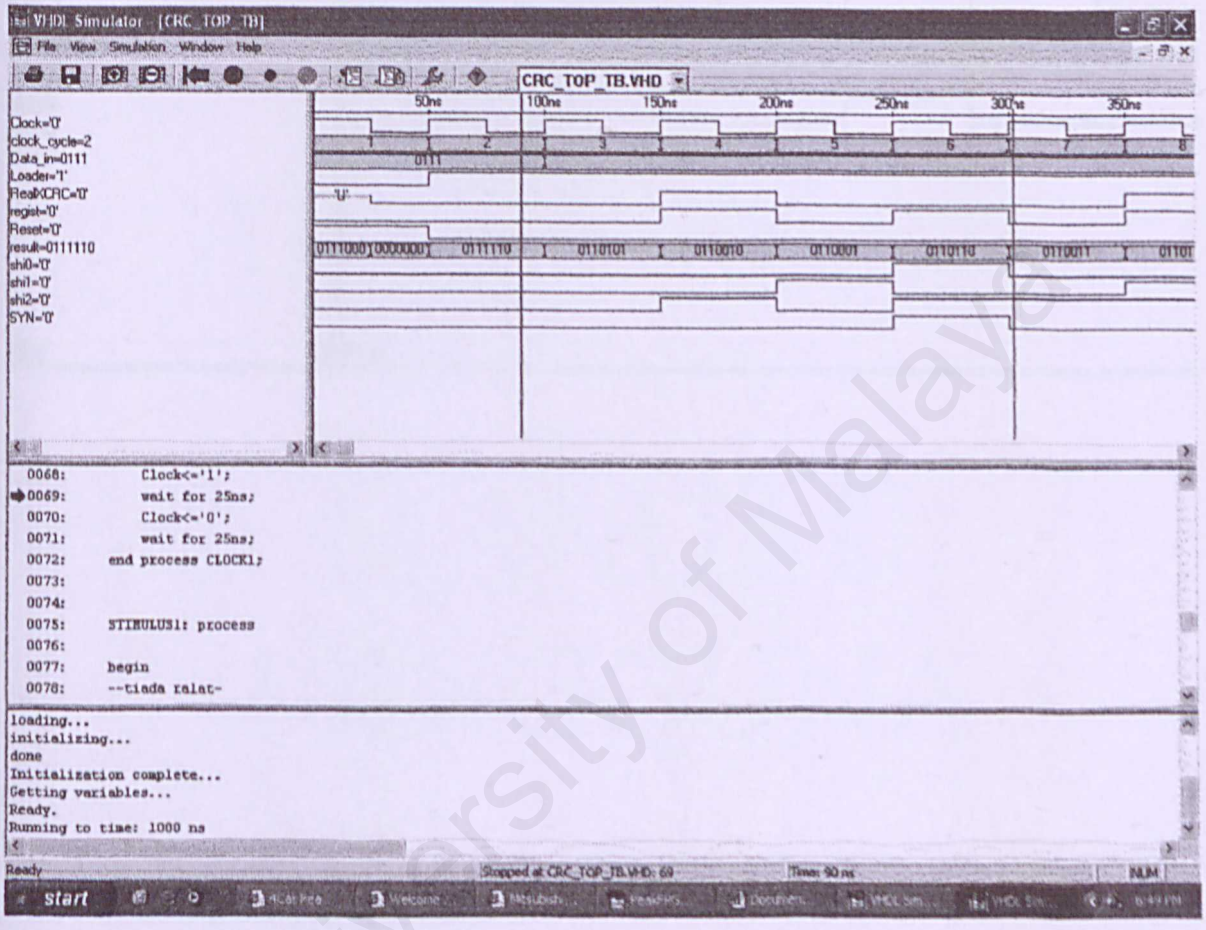
```
  --data_in<='0';
  --rst<='0';
  --wait for period;--Wait One clock cycle


     wait; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

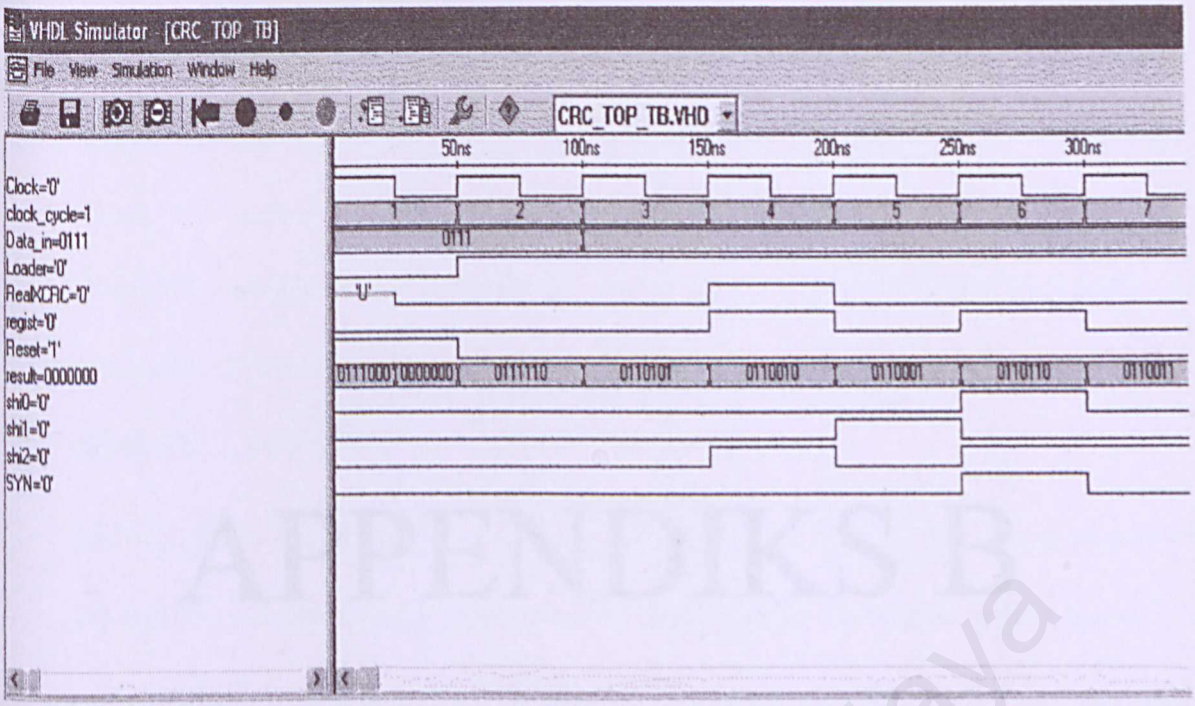
--- CODING UNTUK PD---

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity pd is
    port ( rst, clk_pll, clk_48 : in std_logic;
           rx_active :            in std_logic;
           data_rx_in_valid :     in std_logic;
           data_rx_in :           in std_logic_vector(7 downto 0);
           data_rx_out :           out std_logic_vector(7 DOWNTO 0);
           data_rx_out_valid :     out std_logic;
           data_rx_done:          out std_logic
           );
end pd;

architecture mixed of pd is

type pd_state is (idle, active, data);

-- Internal signals
signal pid_DATA0_int, pid_DATA1_int : std_logic;
signal state : pd_state;
signal pid : std_logic_vector (7 downto 0);-- Internal register for
datain
signal pid_ld_en : std_logic; -- PID Load enable
signal pid_le : std_logic; -- PID load control
signal pid_err: std_logic;
signal pid_data : std_logic; -- What type of packet was recieved
signal data_valid_int, data_done_int : std_logic; -- Data Packet load
control



begin

-- Recieved PIDs
pid_DATA0_int<='1'  when pid(3 downto 0)= "0011" else '0';
pid_DATA1_int<='1'  when pid(3 downto 0)= "1011" else '0';

--Decode type of PID
pid_data <= (pid_DATA0_int or pid_DATA1_int);


--PID load enable logic
pid_ld_en <= (data_rx_in_valid and pid_le and rx_active);

-- Latch data from NRZI decoder
p1: process(clk_pll, rst)
       begin
          if rst ='1' then
             pid<="11110000";
          elsif rising_edge(clk_pll) then
```

```
            if pid_ld_en ='1' then
               pid<= data_rx_in;
            end if;
         end if;
end process;

-- PID check and error generating
pid_err<= '1' when pid(3 downto 0) /= not pid(7 downto 4)
          else '0';


-- Output data
p4: process(clk_pll, rst)
         begin
         if rst='1' then
            data_rx_out <= (others=>'0');
            data_rx_out_valid<='0';
         elsif rising_edge(clk_pll) then
               if (data_valid_int and data_rx_in_valid) ='1' then
                  data_rx_out <= data_rx_in;
                  data_rx_out_valid<='1';
               else data_rx_out_valid <='0';
               end if;
         end if;
end process;
data_rx_done <= data_done_int;

-- Control state mchine
sm: process(clk_pll, rst)
      begin
         if rst ='1' then
            state <= idle;
            data_valid_int <='0';
            data_done_int <='0';
         elsif rising_edge(clk_pll) then
            case state is
               when idle =>
                  data_valid_int <='0';
                  data_done_int <='0';
                  if (data_rx_in_valid and rx_active) ='1' then
                     state <= active;
                  end if;
               when active =>
                  if (pid_data and(not pid_err))='1' then
                     state <= data;
                     data_valid_int<='1';
                  elsif rx_active='0' then
                     state <= idle;
                  end if;
               when data =>
                  if rx_active ='0' then
                     data_done_int <= '1';
                     state <= idle;
                  end if;
               when others =>
                  state <= idle;
            end case;
```

```
      end if;
end process;

--PID latch control signal
with state select
      pid_le <='1' when idle,
               '0' when others;

end mixed;
```





APPENDIKS B

MANUAL PENGGUNA

About Accolade

Founded in Redmond, Washington in 1994, Accolade Design Automation, Inc forged a strong reputation as a systems integrator and software development firm offering advanced tools for field programmable gate array (FPGA) and system level design, with particular emphasis on hardware description language (HDL)-based design. PEAKFPGA was one of Accolade's key products.

These steps will introduce user to the features of PeakFPGA by taking you step-by-step through the simulation and synthesis of an sample project included in the product. The sample project we've used is a controller for a video frame capture unit that has been implemented in VHDL as a state machine.

The steps within this tutorial are as follows:

Step 1: Open the sample project

Step 2: Prepare the project for simulation

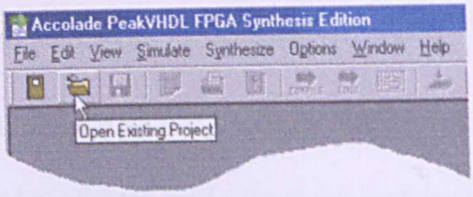
Step 3: Simulate the project

Step 4: synthesize to an FPGA

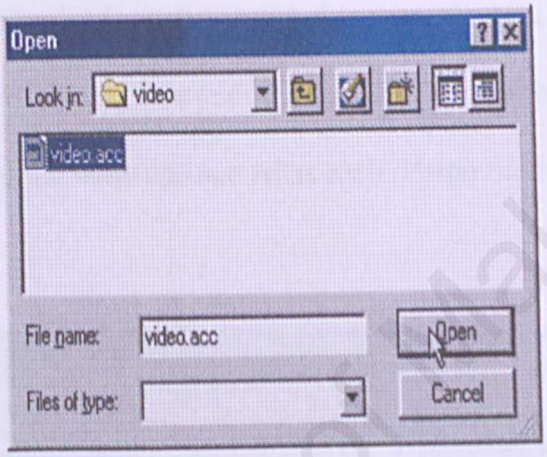
Use the Quick Jump menus at the head and foot of each page to skip through the topics.

Step 1: Open the sample project

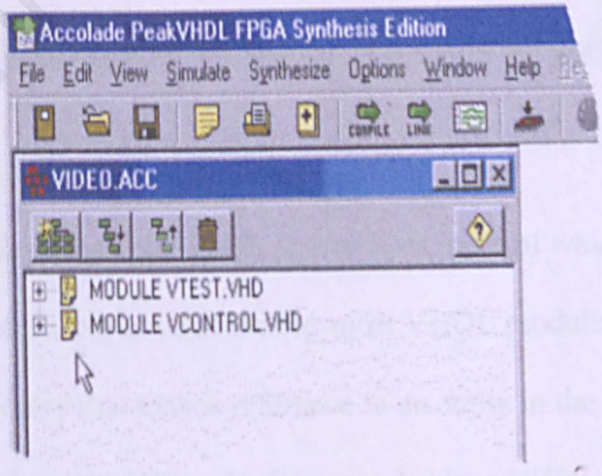
Begin by launching the PeakFPGA application and clicking on the **Open Existing Project** toolbar button as shown below:



Navigate to the **examples** directory and choose the **VIDEO.ACC** file from the **Video** subdirectory as shown below:

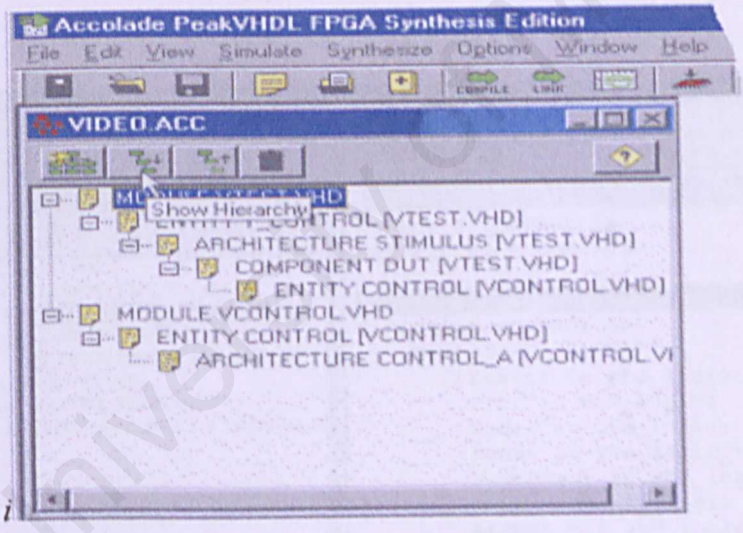


When the project is open, you will see the Hierarchy Browser child window, listing the two VHDL source files associated with this project:



The Hierarchy Browser is the place where you will select files for editing, invoke processes for simulation and synthesis, and otherwise manage your design files. The Hierarchy Browser also gives you valuable information about the structure of your VHDL design, such as the relationship of lower-level VHDL design units (entities, architectures, components, etc.).

To view the complete hierarchy for your design, for example, you can select the **Show Hierarchy** button in the Hierarchy Browser toolbar (or click on the small + icons next to each file name) to expand the view and see what each design module (VHDL source file) is composed of:

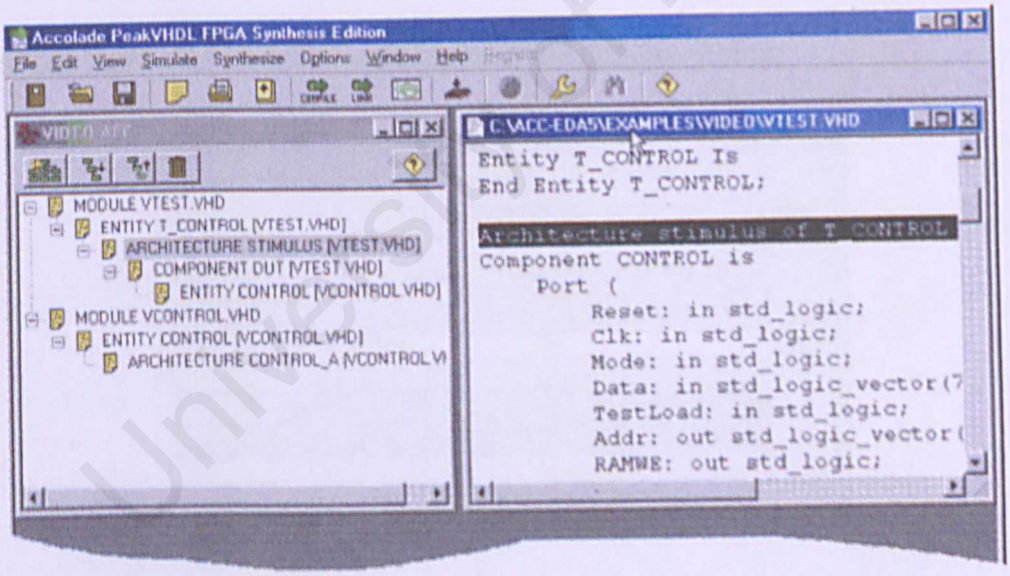


In this simple project there are two VHDL source files, each of which includes one entity and one architecture. Notice that the top-most VHDL module (**VTEST.VHD**) also includes a component entry that makes reference to an entity in the second module, **VCONTROL**. This is how the Hierarchy Browser displays VHDL hierarchy

information, and how it determines file dependencies (and order of compilation) when processing your designs.

Note: the order in which VHDL modules are included in a given PeakFPGA project are not significant. You may choose to have top-level VHDL source files (such as test benches) at the top of the Browser list as shown in this example, or you many wish to have them listed in some other order. PeakFPGA allows you to re-order modules within a project at any time to suit your needs.

To edit a VHDL source file, simply double-click on the desired module name, or on any lower-level design unit name (such as an entity or architecture name) in the Hierarchy Browser. Double-clicking on a name in the Browser invokes the built-in text editor as shown below:



Note that in this example we have double-clicked on the entry for **ARCHITECTURE STIMULUS**, and the editor has jumped to the corresponding section of VHDL code.

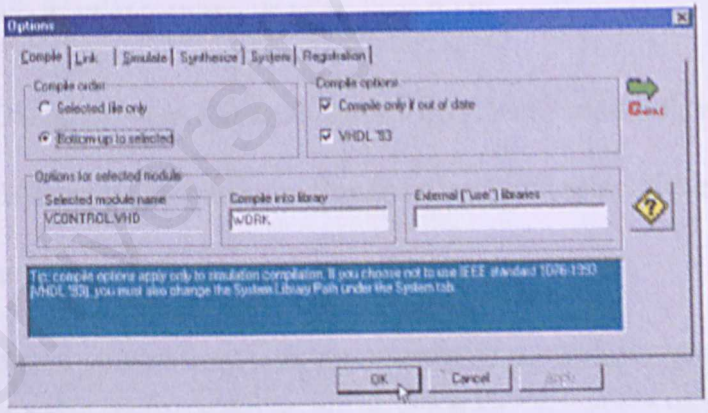
Note: PeakFPGA allows you to specify an external text editor to be invoked when project files are edited. Refer to the PeakFPGA help files for more information.

Step 2: Prepare the project for simulation

The first step in processing this design is to prepare it for simulation. This involves two steps: first compiling each of the source files, and then linking the resulting compiled output files together to create a simulation executable.

PeakFPGA can, if desired, perform these steps automatically (using the dependency checking features of the Hierarchy Browser) each time you invoke the simulator. For illustrative purposes, however, we will compile each file in this sample project individually.

Before we compile this design for simulation, let's first look at the compiler options available. To see the compile options, select the **Options** menu or click on the **Options** icon (the one that looks like a wrench) to open the Options dialog:



The **Compile** tab (which is the default tab) of the Options dialog shows the options available during compilation. These options are documented in the Help system (directly

accessible via that question mark icon to the right). The options we have selected for this project are:

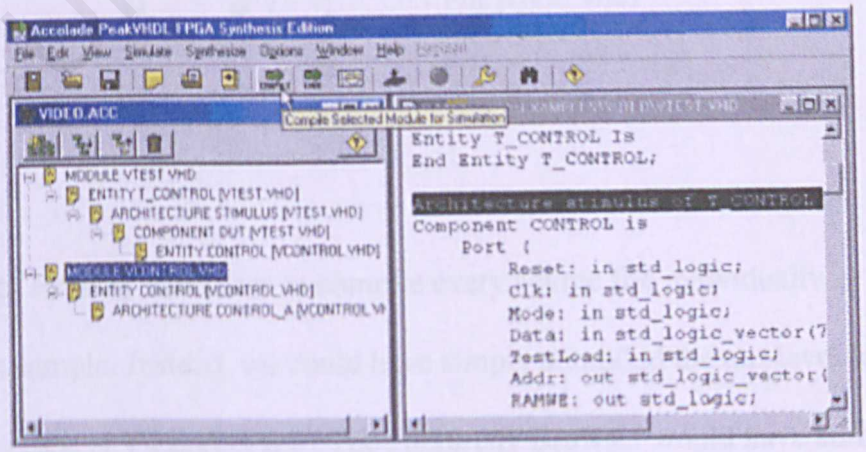
Bottom-up to selected. This option instructs PeakFPGA to look for any lower-level VHDL files (those that the current file depends on) and compile them before compiling the selected file.

Compile only if out of date. This option prevents files from being re-compiled if they are already compiled and up-to-date. This can be a big time saver for larger designs consisting of many source files.

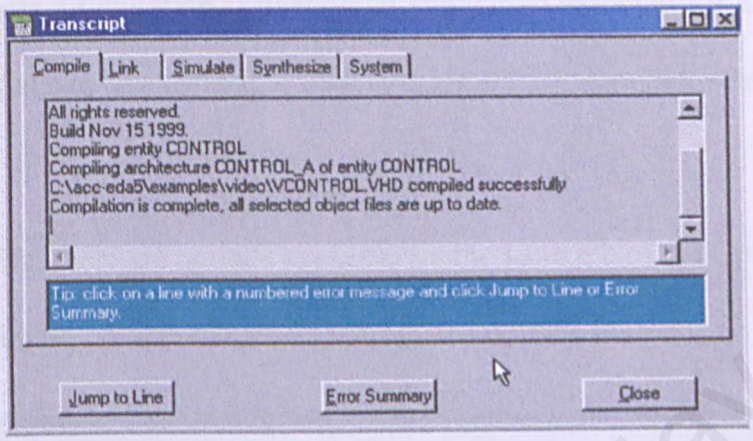
VHDL '93. This option specifies that the compiler should accept the 1993 VHDL standard (IEEE 1076-1993).

Compile into library. This option specifies that the currently highlighted module (the one being compiled) is to be compiled into a library called **WORK**. (As specified by the VHDL language standard, this is the default compile library.)

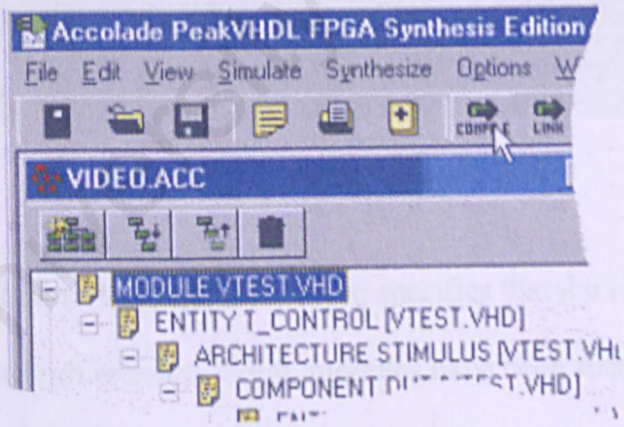
To compile a VHDL module (in this case the **VTEST.VHD** module), you first highlight the module in the Hierarchy Browser (as shown below) and select the **compile** button (or select **Compile** from the **Simulate** menu):



When you start the compile process, a transcript window appears and displays status messages, as well as reporting syntax or other errors found in your source file:



There were no errors in this sample source file, so we repeat the process by selecting and compiling the **VTEST.VHD** module

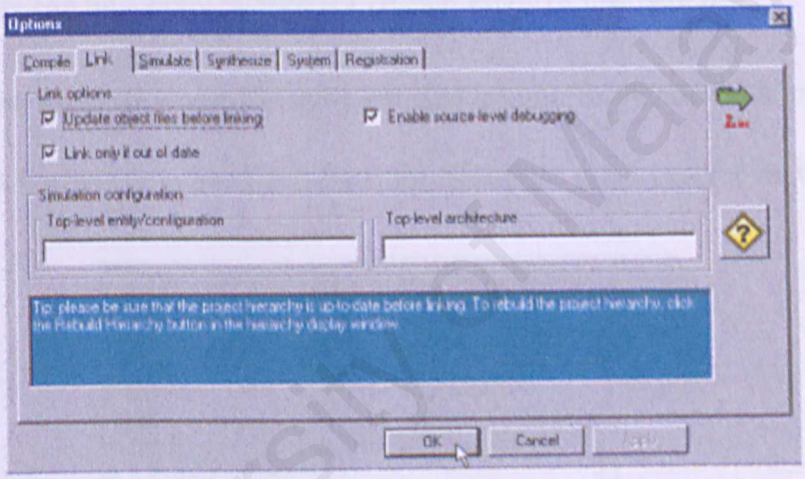


Note: it is not actually necessary to compile every source file individually as we are doing in this sample. Instead, we could have simply compiled the top-level module, which in this case is **VTEST.VHD**. The Hierarchy Browser would have automatically

compiled the lower-level **VCONTROL.VHD** module first if it were found to be out of date.

The next step in preparing the project for simulation is to link together the two compiled modules and create a *simulation executable*. A simulation executable is a special kind of executable file that can be executed in PeakFPGA's VHDL simulation and debugging environment.

As with compiling, there are options available (in the **Link** tab of the Options dialog) that control the linking process. For this sample, the options set are:



Update object files before linking. This option specifies that the Hierarchy Browser will check to make sure all relevant VHDL modules have been successfully compiled before starting the link process. When this option is selected, it is not necessary to manually compile the VHDL source files (as we did in the previous step) before invoking the Link process.

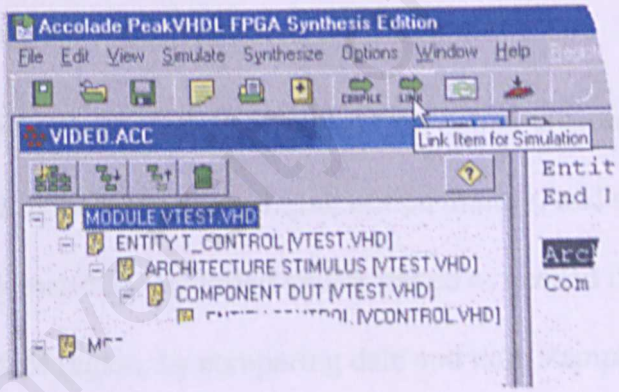
Link only if out of date. This option (which is similar to the **Compile only if out of date** option discussed previously) prevents the link process from being run if the

simulation executable is already up to date as indicated by its time stamp. (If the simulation executable is new than all VHDL files and object files that it depends upon, it will not be re-linked.)

Enable source-level debugging. This option instructs the code generation software (the portion of the linker that actually generates Windows executable code) to insert additional information to allow debugging of the design at the source code level.

Simulation configuration. These two text entry fields (which are left blank in this example) allow you to re-specify the default top-level entity and architecture used for simulation. This can be a convenience for certain kinds of test benches.

To start the Link process, select the top-level design unit (or the top-level module) and select the **Link** button (or select **Link** from the **Simulate** menu) as shown below:

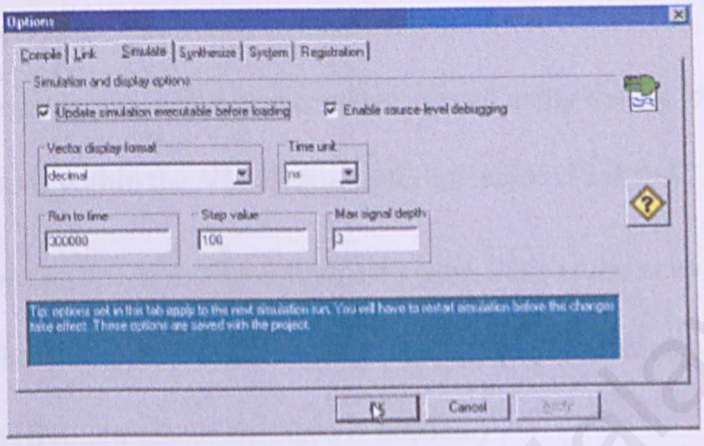


The transcript will again appear and the Link process will execute. Errors (if any) will be reported to the transcript.

The project is now compiled, linked and ready for simulation.

Step 3: Simulate the project

Before launching the integrated simulator, let's first look at the simulation options that have been set for this project. To view or change the simulation options, we open the **Simulate** tab of the Options dialog as shown below:



Note that the following options have been set:

Update simulation executable before loading. This option is closely related to the similar options found in the Compile and Link option dialogs, and specifies that the project should be automatically compiled and/or linked as needed if the simulation executable is out of date (again, by comparing date and time stamps of the VHDL source files and subsequent compiler and linker output files). When this option is set, it is not actually necessary to manually compile or link the project: PeakFPGA will determine the files to be compiled and will prepare the project for simulation automatically.

Enable source-level debugging. This option is related to a similar option in the Link Options dialog, and simply instructs the VHDL simulator to display a source-level debugging window upon loading.

Vector display format. This option specifies how multi-bit VHDL values (vectors) are to be displayed. Valid options are **binary**, **octal**, **decimal** and **hexadecimal**.

Time unit. This option specifies the default time unit to be displayed in simulation waveforms.

Run to time. This option specifies the default end time for simulation. This is the time value to which the simulator will advance when started using the **Go** button. (This value can be changed from within the VHDL simulator interface if needed.)

Step time. This option specifies the default step time. This is the amount of time that the simulator will advance when the **Step** button is selected. (This value can be changed from within the VHDL simulator interface if needed.)

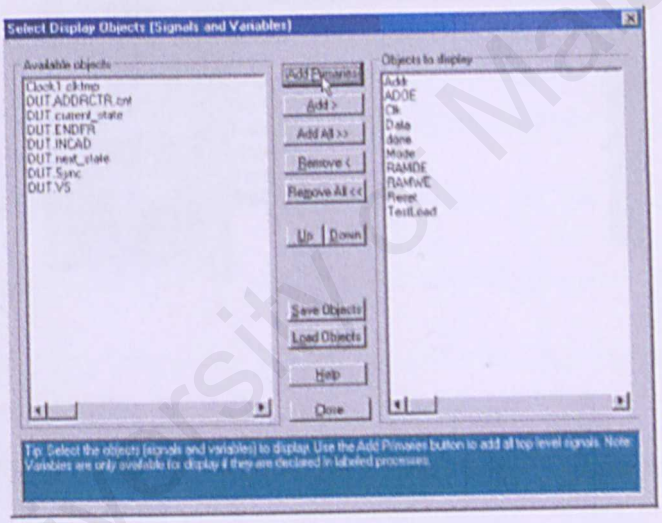
Max signal depth. This option allows you to control how deeply in the hierarchy the simulator should go when extracting and displaying signals for possible viewing. For simulation models such as post-route models, this option can speed the time needed for the simulator to load your design.

Note: the options that you specify in the Compile, Link and Simulate option dialogs are saved with your project so there is no need to re-specify them every time you reopen the project.

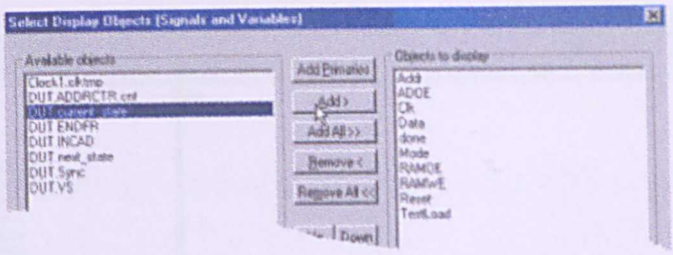
When you have finished reviewing and/or changing the simulation options, you start the simulator by selecting the **Load Simulation** button from the toolbar, or by selecting **Load Simulation** from the **Simulate** menu. Immediately after selecting this function, the VHDL simulator application window appears and the following dialog is displayed:

This dialog (the Select Display Objects dialog) allows you to specify which of the objects (ports, signals and certain variables) in your design you wish to include in the waveform display.

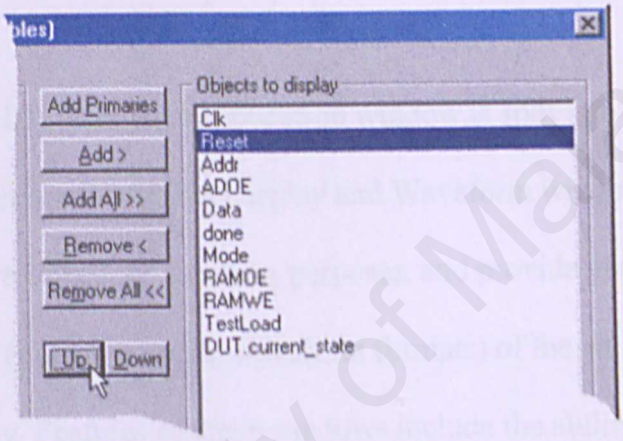
The list on the left side of the dialog shows you what objects are currently available for display. Various buttons are provided for selecting objects and arranging them in the display order desired. The **Add Primaries** button, for example, will select all signals found at the highest level in your design. (These are typically the signals you have declared in your test bench.) We'll use the **Add Primaries** button to include these signals now:



If there are other objects lower in the design hierarchy that you would like to display, you can select them individually using the **Add** button. For this project, we will select a lower-level signal named **DUT.current_state** and add it to the display:

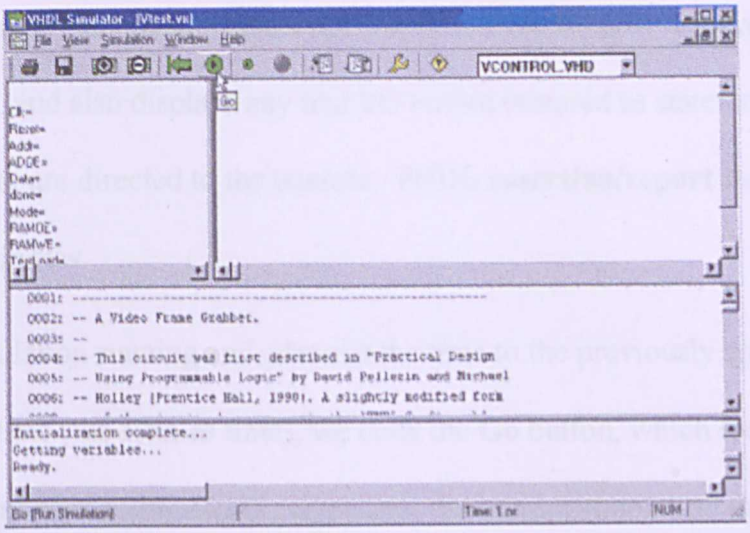


Finally, we will select individual objects from the **Objects to display** list and reorder them (for easier debugging) using the **Up** and **Down** buttons:



*Note: you only need to select and order the signals once for a given project. By default, the simulator remembers your most recent signal selections and ordering, and if needed you can also use the **Save Objects** and **Load Objects** feature to save specific collections of object names and orderings.*

When we have selected and ordered the objects, we close the dialog using the **Close** button. The selected objects are added to the VHDL Simulator display as shown below:



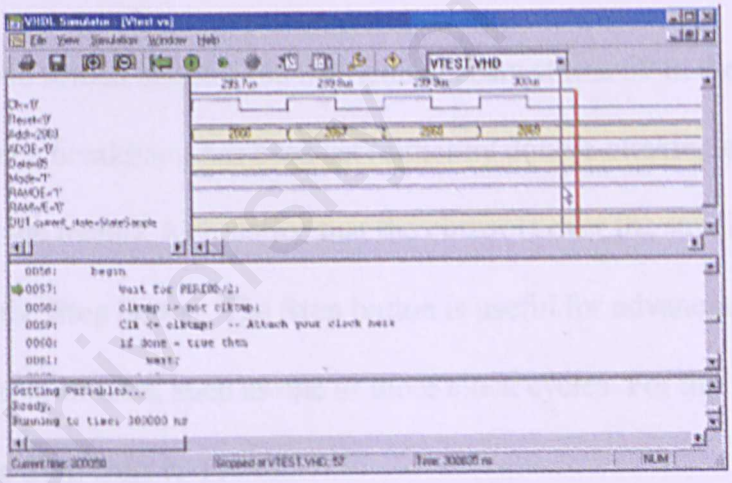
As you can see, the VHDL simulator application window is split into four distinct panes. The top-most two panes are the Object Display and Waveform windows, respectively. These two windows are linked for scrolling purposes, and provide you with an up-to-date view of the value (in both text and waveform formats) of the value of each object you selected previously. Features of these windows include the ability to zoom and pan through the waveform, drop measurement cursors and rearrange objects (in terms of their vertical display order) at will.

The center window is the Source Code Display window. This window shows the current line of code that is being executed in your VHDL design. This window is also where you set breakpoints. A drop-down list of source files (located up in the toolbar) allows you to select alternate source files for setting of breakpoints. Single-step features (also controlled from the toolbar) let you advance simulation one line at a time and see the results in the Source Code Display and Waveform windows.

The bottom-most window is the Transcript window. This window displays simulator status messages, and also displays any text I/O output (entered as statements in your VHDL code) that are directed to the console. VHDL **assertion/report** statements also appear in this window.

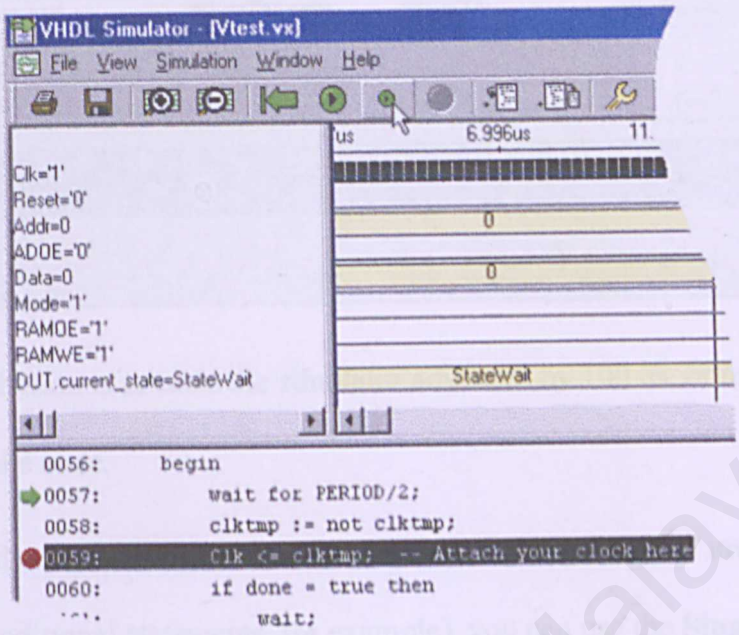
To start the simulation running and advance the time to the previously specified simulation end time (the **Run to** time), we click the **Go** button, which looks like a green VCR play button. As the simulation progresses, the current simulation time is displayed in the status bar at the bottom of the application frame, and status messages (if any) are displayed in the Transcript window.

After the simulation has finished, a waveform showing the simulation results is displayed as shown below:



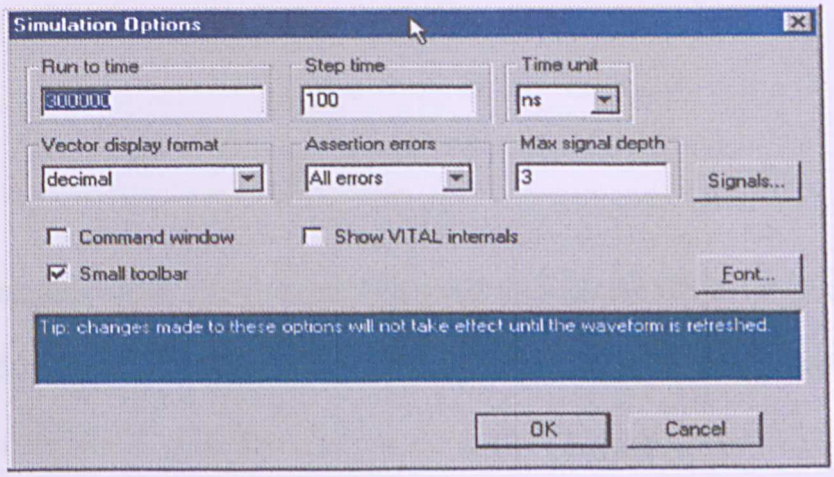
At this point you may be satisfied with the results of simulation (as shown in the Waveform window), or you may need to investigate the results further. The VHDL simulator has various features allowing more detailed analysis. For example, you can

use the Source Code Display window to set one or more breakpoints in the code to investigate the control flow through your design. This is shown below:



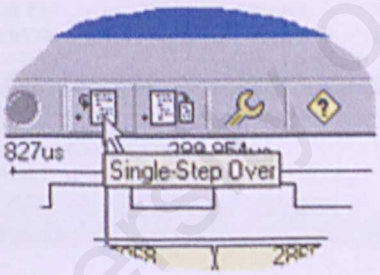
Notice in the above screen image a red ball icon appears at line 59 in the source file. This indicates that a breakpoint has been set (either by double-clicking on that line in the window or using the menu). Also notice that the cursor is over the smaller green start button, which is the **Step** button. The **Step** button is useful for advancing simulation by a predetermined about of time, such as one or more clock cycles. For this project, we have previously set the **Step time** to 100 ns.

The **Step time** value and other simulation values can be changed at any time by selecting the Options dialog from within the VHDL simulator as shown below:

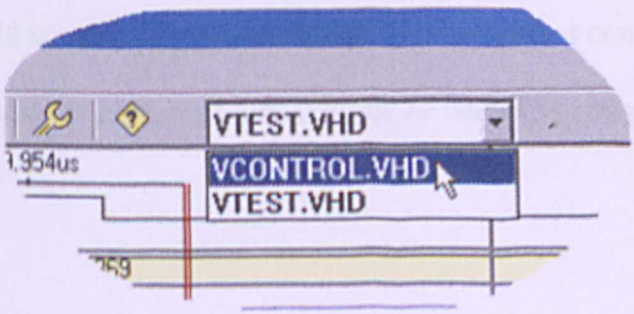


When the **Step** button is pressed the simulator advances by 100 ns, or until it encounters a breakpoint in the code.

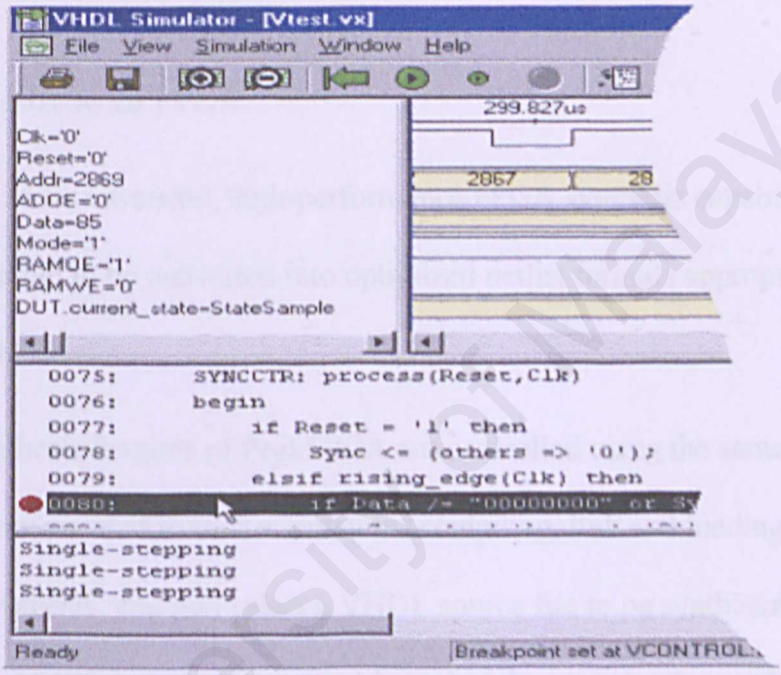
If you wish to advance the simulator one source file line at a time (to investigate control flow through conditional statements, for example), you can use the **Single-Step** buttons (**Step Over** and/or **Step Into**) as shown :



Breakpoints can be set at any executable line of VHDL code. If you need to set a breakpoint in a file other than the one currently displayed, you will use the drop-down list of file names to select the file as shown :



After selecting the proper file, you can then scroll through the file in the Source Code Display window and set a breakpoint at the desired line. In this example we'll set a breakpoint in the VHDL code that controls a 0 to 127 (7-bit) counter:



When we next advance the simulation (either by increasing the **Run to time** value and selecting the **Go** button or by using the **Step** button), the simulator will stop at the selected breakpoint (assuming that line of VHDL code is in the execution path).

At this point we could continue single-stepping to verify correct control flow through the counter logic, we could set additional breakpoints, or we could compare the results of

simulation (the values seen on various signals and variables) against our expectations to track down logic problems.

The VHDL simulator has other features that are documented in the on-line help. For now, however, let's assume that this design is working as expected and move on to the FPGA synthesis phase of the project. You can close the VHDL simulator application window at this point, or simply minimize it.

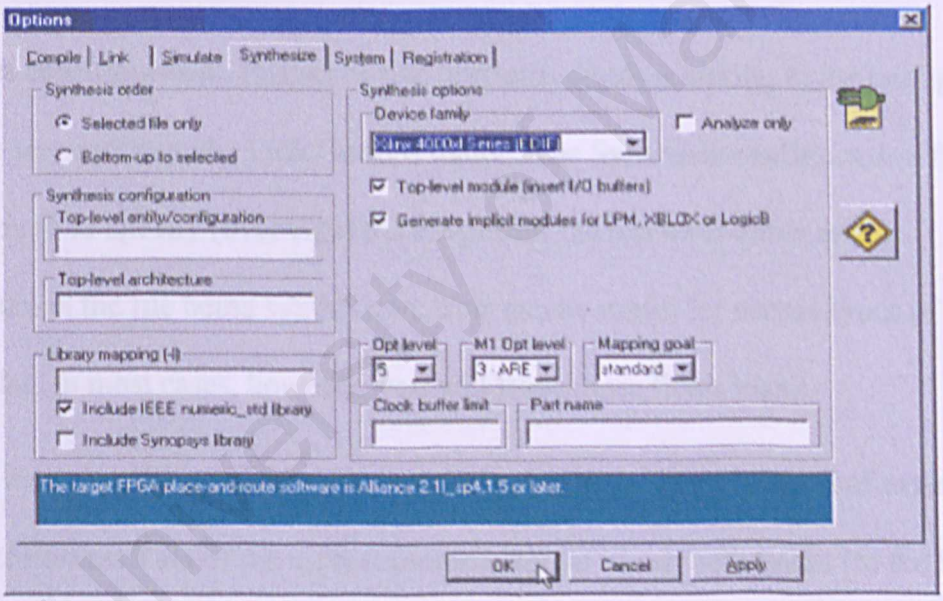
Step 4: Synthesize to an FPGA

PeakFPGA includes advanced, high-performance FPGA synthesis capabilities allowing your VHDL design to be converted into optimized netlist formats appropriate for a wide variety of FPGA devices.

The FPGA synthesis features of PeakFPGA are controlled using the same Hierarchy Browser window we used to start simulation (compiling, link and loading) earlier in this tutorial. For synthesis, you will select a VHDL source file to be synthesized (which may reference other VHDL source files using hierarchy) then invoke the synthesis compiler by selecting the **Synthesize Selected Module** button.

Before selecting this button, however, let's take a moment to review the synthesis options and select a specific type of FPGA device.

To view and change synthesis options, we first open the Options dialog and select the **Synthesize** tab (or select **Synthesize Options** from the **Options** menu):



For this project we have selected the following options:

Selected file only. This option specifies that we will be synthesizing only the currently selected source file. In this sample project there is only one synthesizable VHDL file (the top-level file is the test bench, and is not synthesizable), but in many larger designs you will have multiple synthesizable files. In designs like that, you may wish to combine all the synthesizable files by selecting the top-most synthesizable file and letting PeakFPGA determine the lower-level files to be included based on their hierarchical relationships. At other times (depending on the capabilities of the FPGA place-and-route software you are using) you might prefer to synthesize each file individually and merge the resulting netlist files later in the process. PeakFPGA supports both methods of design.

Synthesis configuration. This set of two text entry fields is similar to the configuration fields we saw earlier in the Link Options dialog. The **Synthesis configuration** field allows you to re-specify (over-ride) the default for the top-level entity and/or architecture in the file being synthesized. This can be useful for certain types of complex design files. In most cases, however, you will leave these fields blank.

Library mapping. These fields give you control over the interpretation of external library references. Two of the most common external library references (to the IEEE standard **numeric_std** and Synopsys **std_logic_arith** libraries) are given simple check boxes to simplify their use. Other non-standard library references can be entered in the text box provided. The on-line help describes this feature in more detail.

Synthesis options. The collection of option settings begins with the **Device family** option, a drop-down list showing all supported FPGA families. In this example we have

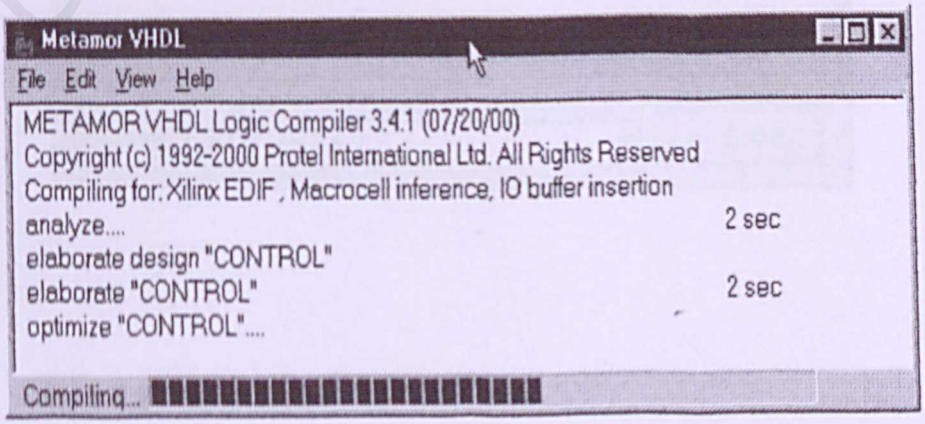
chosen the **Xilinx 4000xl** series of devices. (For a complete list of supported device families, refer to the Synthesis Release Notes or to the Protel web site.)

Synthesis options appearing under the **Synthesis options** section are FPGA device-dependent (with the exception of **Analyze only**, which allows you to turn off final optimization and netlist output for the purpose of synthesis checking only). Choosing a different device family from the drop-down list will result in a different set of synthesis options appearing. Describing each possible option is beyond the scope of this tutorial; the options are fully described in the on-line help.

Also notice in the previous screen image that the teal-colored tip text in the lower portion of the dialog is also device-specific. This tip text provides useful information, such as the specific FPGA place-and-route software (and minimum version number) you will need to fully process your design.

When you have finished selecting synthesis options you will select the **Close** button to exit the Options dialog.

When you are ready to start synthesis, select the desired module in the Hierarchy Browser (in this case **VCONTROL.VHD**) and select the **Synthesize** button. A synthesis transcript appears as shown below:

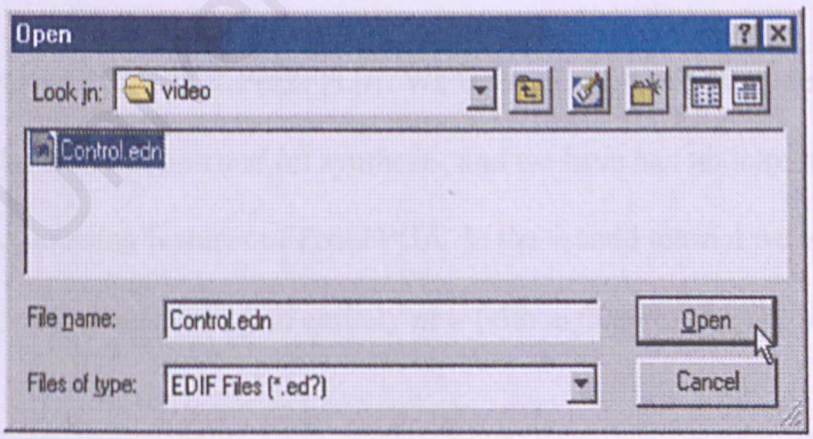


This transcript window shows the status of synthesis as it is running. If there were any errors in the design (such as the use of un-synthesizable language constructs), these errors would be displayed and the transcript would remain on the screen until you closed it. If there are no errors, the transcript automatically closes.

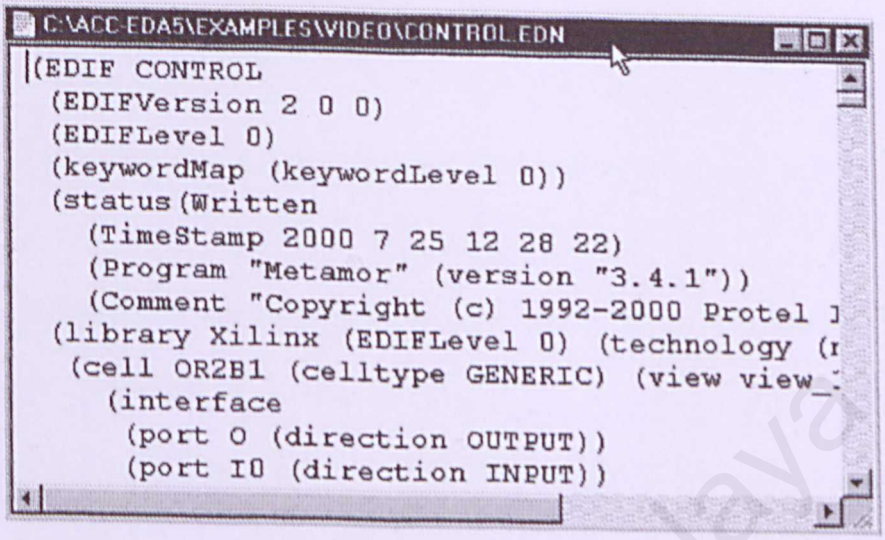
You can view the transcript text after the synthesis transcript window is closed by selecting **View Synthesis Results** from the **Synthesize** menu.

The transcript text available from **View Synthesis Results** is somewhat more detailed than that appearing in the transcript window, and includes information about the device resources (registers and higher-level macrocells) that will be used by your design. The transcript report also gives a brief summary of device utilization (number of logic blocks) that can help you in selecting a specific device from the family you are targeting.

If you wish to view the actual results of synthesis (in the form of the generated netlist), you can select the **Open File** toolbar button (or select the equivalent item from the **File** menu) as shown below:



The generated netlist (which in this case is EDIF format) is loaded into the built-in text editor:



Your design is now converted to a netlist and is ready for placement and routing using the tools provided by your FPGA vendor.

Summary

This concludes our first tour of PeakFPGA. We have seen how a VHDL design can be processed for both simulation and for synthesis, and we have had an introduction to most of the design processing features of PeakFPGA. In the second tutorial we will focus more on design entry by creating an entirely new (although quite simple) VHDL project.

RUJUKAN

Rujukan

[1]. "Data Communication and Networking" Behrouz A.Forouzan, MC Graw Hill 2nd Edition

[2]. "The Intel Microprocessor" Berry B. Brey, Prentice Hall 6th Edition

[3]. http:// www.howstuffwork.com

[4].http://feynman.ee.ualberta.ca/~elliott/ee552/projects/2001f/Where's_Waldo_image_finder/final_rep.pdf

[5]. http://www.willas-array.com/prod/products/directory/pdf/genesys/GL841.pdf

[6]. http://www.xilinx.com/esp/dut/cdu/collateral/usb20.pdf

[7]. http://usb.org/developers/whitepapers/crodes.pdf

[8]. http://www.usbstuff.com/

[9]. http://www.xess.com/

[10]. http://www.beyondlogic.org/usb/usbhard2.html

[11]. http://www.fairchildsemi.com/an/AN/AN-5015.pdf

[12]. http://www.lowyatt.com.my

[13]. http://computer.howstuffworks.com/usb7.htm

[14].http://www.tech-pro.net/intro.usb.html

[15].http://www.jaycar.com.au/images_uploaded/usbbus.pdf